AFRL-IF-RS-TR-2002-305
Final Technical Report
December 2002

QUORUM DISTRIBUTED OBJECT INTEGRATION (QUOIN)

BBN Technologies

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J098

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-305 has been reviewed and is approved for publication.
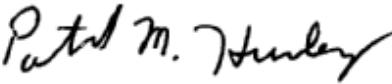
APPROVED: *Patrick M. Hurley*
PATRICK M. HURLEY
Project Engineer

FOR THE DIRECTOR: *signature*
WARREN H. DEBANY, Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE DECEMBER 2002	3. REPORT TYPE AND DATES COVERED Final JUN 98 – DEC 02

4. TITLE AND SUBTITLE
QUORUM DISTRIBUTED OBJECT INTEGRATION (QUOIN)

5. FUNDING NUMBERS
C   - F30602-98-C-0187
PE - 62302E
PR - G116
TA - 00
WU - 01

6. AUTHOR(S)
Richard E. Schantz, Joseph Loyall, Partha Pal, Michael Atighetchi, Franklin Webber, Richard Shapiro, Craig Rodrigues, John Zinky, Douglas Schmidt, Christopher Gill, William Sanders, Michael Cukier, Irfan Pyarali, Yamuna Krishnamurthy

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
BBN Technologies
10 Moulton Street
Cambridge Massachusetts 02138

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency   AFRL/IFGA
3701 North Fairfax Drive   525 Brooks Road
Arlington Virginia 22203-1714   Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2002-305

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Patrick M. Hurley/IFGA/(315) 330-3624/ Patrick.Hurley@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

Complex computing systems are predominantly networked. As a result, they must operate under increasingly unpredictable and changeable conditions. This problem occurs at multiple levels, ranging from distributed applications that are geographically dispersed to applications that are embedded and componentized. Both types of applications pose similar problems and issues, and lend themselves to similar solutions using strategies, algorithms, and optimizations at various levels of granularity. In particular, they require predictable mission-critical levels of service end-to-end. Adaptive distributed object computing (DOC) middleware is a promising approach to provide these end-to-end services by coordinating lower-level mechanisms that coordinate application- or user-centric tradeoffs.

 However, the current generation of commercial-off-the-shelf (COTS) middleware lacks adequate support for real-time applications with stringent quality of service (QoS) requirements. Moreover, there is a pressing need to coordinate individual advances in the solution space being addressed by different parts of the research community. The problems faced by distributed and embedded real-time application developers are particularly challenging, with many interlocking aspects. Unless pieces of the emerging solution are available as coordinated, integrated packages, the value of middleware is severely diminished and may in fact make matters worse instead of better, thereby being unacceptable to system developers and integrators.

14. SUBJECT TERMS
Quality of Service, Adaptive Resource Management, Distributed Object Computing, Middleware, Real-Time

15. NUMBER OF PAGES
120

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Table of Contents

List of Figures and Tables

1. Overview

This document is the final report for QuOIN, the Quorum distributed Object Integration project. This DARPA project, with additional funding from the Air Force Research Laboratory, ran from June 1998 through December 2001, under contract No. F30602-98-C-0187. During its lifetime, QuOIN focused on three separate technology aspects:

- An architecture and design for a multi-layer, middleware based model and implementation for providing and managing adaptive end-to-end QoS in a distributed object applications,
- An integration of a variety of Quorum technologies into this architecture and design as a proof of concept for a number of specific QoS dimensions,
- Test, evaluation and transition activities using the proof of concept implementation on real world, domain specific problems, providing leave behind Quorum products in the application domain, and providing useful feedback on the continuing design and implementation.

BBN Technologies (BBN) was the prime contractor for the QuOIN project. Washington University in St. Louis (WUSTL), the University of California, Irvine (UCI), the University of Illinois, and OOMWorks Inc. were subcontractors to BBN for this effort. In addition, a number of other Quorum program participants have integrated their ideas and technologies with the evolving QuOIN software as part of the Quorum integration activities, and still other Quorum participants have participated in QuOIN transition, test and evaluation activities.

The Quorum program is revolutionizing the way that people approach network-centric computing in the 21st century. The primary element in Quorum's strategy of transformation is focusing on middleware based architectures built around adaptive approaches to Quality of Service (QoS).

The Quorum Distributed Object Integration (QuOIN) project was a key component of the Quorum program by building advanced middleware that introduces adaptive QoS capabilities into the distributed object computing (DOC) software development model. This new QoS-enhanced DOC middleware is based on a combination of existing commercial-off-the-shelf (COTS) software and emerging technology currently under development by the Quorum research community. By integrating these software capabilities, QuOIN has produced a DOC environment with emerging managed QoS properties in the following areas:

- Reserved/managed network bandwidth
- Availability
- Real-time behavior
- Security (access control)

Highlights of the results from the recently concluded QuOIN activity include:

1. An architecture for integrated, end-to-end adaptive QoS
2. Detailed design and implementation of a middleware based service layer for constructing adaptive applications (QuO)
3. Detailed design and implementation of a QoS property (dependability) within the evolving adaptive QoS architecture (AquA)
4. Progress in the evolution of a realtime Corba ORB and its services, focused on end-to-end adaptive behavior

5. Progress on adaptive management of security and network bandwidth
6. 3 major releases of the QuOIN software to the Quorum community, available as open source
7. Cooperative development of two exemplar applications as concept demonstrations and technology transition vehicles
8. Organizing and carrying out a series of Quorum working groups as a means of focusing the various Principal Investigator participants around technical topic areas of interest
9. Demonstrations of the technology and the transition applications for technologists at PI meetings, adopters at Quorum technology forums and DARPATech and for users as part of transition activities
10. Numerous journal and conference papers, presentations and theses to promote the ideas and educational dissemination objectives of the program

The most significant summarizing result from all of these (and related) activities is that the adaptive, integrated middleware based approach to building network-centric, responsive systems is now firmly entrenched in the technical community as a cohesive technical research agenda, and initial parts of the required technology base are already being transitioned into real use cases. Acceptance and early adoption has been most visible in those (DoD) application domains involving precision and rapid reaction in volatile, mobile distributed environments.

This report summarizes the activities, progress made and impact of the QuOIN project on its technical and transition agenda, as well as new visions for carrying this work forward. It is organized as follows: Section 2 presents an overview of Quorum and the architecture, design and technologies/capabilities which formed the basis for QuOIN, including references to additional sources of information about these technologies and examples of applications constructed with the emerging technology base which serve as the vehicles for technical transition activities. Section 3 presents a detailed overview of a dependability architecture and implementation (called AQuA) which has been integrated into the QuOIN family of projects, first as a companion Quorum activity to QuOIN, and then as a subdevelopment of QuOIN itself. Section 4 describes experimentation with standardized middleware mechanisms for controlling real-time behavior in the context of one of our technology transition vehicles. Section 5 provides a synopsis of the 3 major releases of the QuOIN software as open source release to the community. Section 6 presents a detailed plan for taking these ideas further in future R&D activities. Section 7 provides a chronological review of project activities during its lifetime. Section 8 provides a selected list of publications, presentations, and public demonstrations related to or developed under QuOIN.

2. Quorum and QuOIN Technical Summary

Quorum: Toward Middleware-centric Adaptive Quality of Service for Distributed and Embedded Real-time Systems

Complex computing systems are predominantly networked. As a result, they must operate under increasingly unpredictable and changeable conditions. This problem occurs at multiple levels, ranging from distributed applications that are geographically dispersed to applications that are embedded and componentized. Both types of applications pose similar problems and issues, and lend themselves to similar solutions using strategies, algorithms, and optimizations at various levels of granularity. In particular, they require predictable mission-critical levels of service end-to-end. Adaptive distributed object computing (DOC) middleware is a promising approach to provide these end-to-end services by coordinating lower-level mechanisms that coordinate application- or user-centric tradeoffs. However, the current generation of commercial-off-the-shelf (COTS) middleware lacks adequate support for real-time applications with stringent quality of service (QoS) requirements. Moreover, there is a pressing need to coordinate individual advances in the solution space being addressed by different parts of the research community. The problems faced by distributed and embedded real-time application developers are particularly challenging, with many interlocking aspects. Unless pieces of the emerging solution are available as coordinated, integrated packages, the value of middleware is severely diminished and may in fact make matters worse instead of better, thereby being unacceptable to system developers and integrators. This section describes how the DARPA Quorum research program is investigating the technologies, issues, and integration activities that must be resolved to realize the goal of adaptive middleware using QoS-enabled frameworks, patterns, and components, and how these technologies are being used in developing real world applications for test and evaluation and transition.

2.1. Introduction

One side-effect of the rapid growth and deployment of highly networked and interconnected distributed and embedded applications is the requirement for more attention to usability, efficiency, predictability, scalability, and control of the communication mechanisms that underly these applications. It is often hard or impossible to predict *a priori* even approximate configurations or workload mixes in environments composed of highly distributed and embedded components. Therefore, potential solutions must be sufficiently flexible to support varying, yet highly predictable, behavior at different times during an application's lifecycle. Quality of Service (QoS) is a widely accepted term that describes a loosely organized collection of activities and technology initiatives designed to improve and control communication-oriented resource management based on mounting R&D experience with distributed and embedded real-time applications and systems.

Providing effective QoS has always been a factor influencing the usability of applications and systems. It is only recently, however, that QoS as a set of named aspects [12, 16] has become the focus as *user-controllable* properties of distributed system infrastructure enhancements. Increased user control is in contrast to the level of service that has been traditionally *fixed*, identically and uncontrollably for all users and application use-cases, during system conception. The recent focus on user-controlled QoS aspects stems from technology advances in historically challenging research areas, such as static

allocation policies established during design time, unsynchronized media streams in distributed multimedia applications, or the lack of assured communication resources for high-priority users during high-demand periods. In general, the focus on QoS aspects has led the computing and communication research community to devise a number of proposed and implemented improvements to commonly available distributed computing infrastructures.

Viewed in their narrowest form, these recent QoS advances address the capabilities and control mechanisms that are available within the network itself. For instance, multimedia research has focused on the problems of synchronizing various streams of media as they traverse the network independently [31, 39], and on reserving necessary communication capacity [38]. However, our experience from developing suites of integrated, networked planning and scheduling applications reveals other key factors that affect QoS, beyond timely communication services. In its broadest sense, therefore, QoS involves a multitude of aspects that transcend the specific functional behavior of a particular distributed or embedded real-time application. Examples of these broader QoS aspects include real-time performance characteristics, dependability, and security, with adaptive "design once, run anywhere" behavior that can adjust to various changing environments and QoS requirements.

A growing class of mission-critical distributed systems, including defense avionics mission computing, commercial aerospace, manufacturing process control systems, and tactical command and control systems, require end-to-end support for flexible and adaptive end-to-end QoS.   In addition, these mission-critical distributed systems typically require seamless interoperability, distribution over multiple nodes, and the sharing of information in support of rapidly organized joint and coalition missions. Moreover, to reduce development cycle time and level of effort, these types of systems are increasingly composed of commercial-off-the-shelf (COTS) assets consisting of diverse capabilities that are often deployed in environments subject to significant change and stress.

Unfortunately, conventional COTS technology emphasizes functional interoperability with virtually no assurance of, or control over, mission-critical QoS aspects, such as timeliness, precision, reliability, and security. Moreover, conventional COTS implementations do not enable acceptable tradeoffs among these QoS aspects. Exacerbating this problem is the fact that much of the commercial world has adopted a model of "distributed" computing based on least-common-denominator desktop operating systems. These trends are particularly problematic because COTS-based distributed systems are becoming *enabling technologies* for projects in competitive industries where deregulation, global competition, and shrinking R&D budgets motivate the need for increased software productivity, quality, and cost-effectiveness. Unfortunately, no single system integrator or solution provider yet supplies technologies that span the full range of infrastructure required to meet the QoS demands of mission-critical distributed and embedded real-time applications and systems.

The forces and trends outlined above motivate the need to construct systems that operate effectively at the intersection of the following attributes: integrated QoS tradeoffs, adaptive behavior, middleware-driven end-to-end behavior and reusable COTS infrastructure. In this paper, we present strategies for organizing these activities and report progress from the DARPA Quorum research program that is exploring adaptive QoS from a COTS infrastructure and middleware perspective.

The remainder of this section is organized as follows. Section 2.2 outlines a solution to the challenges enumerated above based on *adaptive COTS middleware*; Section 2.3 summarizes recent advances in

this direction within the DARPA Quorum research program; Section 2.4 and 2.5 outline Quorum program concepts and the organizing technical themes; Section 2.6 describes the collection of technologies being developed and integrated under Quorum and QuOIN; Section 2.7 describes two examples of adaptive applications that were developed as technology test, evaluation and transition vehicles; Section 2.8 discusses some additional areas of investigation that remain and Section 2.9 provides some concluding remarks.

2.2. Candidate Solution: Adaptive COTS Middleware

Requirements for faster development cycles, decreased effort, and reusable solutions motivate the use of *middleware*. Middleware is software that resides between applications and the underlying operating systems, protocol stacks, and hardware to enable or simplify how these components are connected and interoperate. The role of middleware, such as Sun's Jini [33], Java RMI [37], and EJB [34] frameworks, Microsoft's DCOM [1], and the Object Management Group (OMG) CORBA [20, 19, 18] middleware, is to decrease the cycle-time and level of effort required to develop high-quality, flexible, and interoperable distributed and embedded systems using reusable software infrastructure component services, rather than building systems entirely from scratch for each use. In general, middleware provides the following benefits: (1) it shields software developers from low-level, tedious, and error-prone details, such as socket-level programming [28], (2) it provides a consistent set of higher-level network-oriented abstractions for developing distributed and embedded systems and (3) it amortizes software lifecycle costs by leveraging previous development expertise and capturing implementations of key design patterns [2, 7] in reusable frameworks, rather than rebuilding them manually for each use.

When middleware is commonly available for acquisition or purchase, it *becomes COTS*. While it is possible *in theory* to develop complex systems from scratch, i.e., without using COTS middleware, contemporary economic and organizational constraints, as well as competitive and interoperability pressures, are making it implausible to do so *in practice*. Thus, COTS middleware plays an increasingly strategic role in software intensive, mission-critical distributed and embedded real-time systems, which is why we base our adaptive QoS-centric R&D activities on COTS middleware.

Distributed object computing (DOC) is the most advanced, mature, and field-tested middleware paradigm available today in which to achieve this flexible adaptive behavior. DOC software architectures are composed of relatively autonomous objects that can be distributed or collocated throughout a wide range of networks and interconnects. Clients invoke operations on target objects to perform interactions and functionality needed to achieve application goals. Within the family of distributed object computing models, we are initially focusing activities on CORBA because it is heterogeneous and because it has a well-established, successful, and open standardization process. Moreover, an increasing number of standards-compliant CORBA implementations [29] are now available for use in mission-critical distributed and embedded systems.

Due to constraints on footprint, performance, and weight/power consumption, development of mission-critical distributed and embedded real-time systems has historically lagged behind mainstream software development methodologies by a considerable amount. As a result, these types of software systems are extremely expensive and time-consuming to develop, validate, optimize, deploy, maintain and upgrade. Moreover, they are often so specialized and tightly coupled to their current

5

configurations and operating environments that they cannot be readily adapted to new market opportunities, technology innovations or changes in run-time situational environments.

In addition to the development methodology and system lifecycle constraints mentioned above, designers of mission-critical distributed and embedded real-time systems have historically used relatively static methods when allocating resources to system components, which often possess competing real-time requirements. For instance, flight-qualified avionics mission computing systems [10] have traditionally established the priorities for all resource allocation and scheduling decisions very early in the system lifecycle, i.e., *well* before run-time. Static strategies have been used for mission-critical distributed and embedded real-time applications because (1) system resources were insufficient for more computationally intensive on-line approaches and (2) simplifying analysis and validation was essential to remain on budget and on schedule, particularly when systems were designed from scratch using low-level, proprietary languages, operating systems, interconnects, and software tools.

The next-generation of mission-critical distributed and embedded real-time systems require an increasingly wide range of features, while at the same time minimizing costs. For instance, next-generation avionics mission computing systems [11] must collaborate with remote command and control systems, provide on-demand browsing capabilities for human operators, and flexibly respond to unanticipated situational factors that arise in run-time environments. Moreover, these systems must perform unobtrusively, shielding human operators from unnecessary details, while simultaneously communicating and responding to mission-critical information at an accelerated operational tempo. The characteristics of next-generation systems present QoS requirements for shared resources and workloads that can vary significantly at run-time. In turn, this increases the demands on end-to-end system resource management and control, which make it hard to simultaneously (1) create effective resource managers using traditional statically constrained allocators and schedulers, (2) achieve reasonable resource utilization and (3) meet the individual application needs for tradeoff preferences. In addition, the mission-critical aspects of these systems require that they respond adequately to both anticipated and unanticipated operational changes in their run-time environment and ensure critical components can acquire available resources [8].

Meeting the increasing QoS demands of next-generation real-time systems motivates the need for adaptive middleware-centric abstractions and techniques, such as the automated reconfiguration, layered resource management, and dynamic scheduling techniques being explored in the context of the DARPA Quorum program [4], which we describe in the following section.

2.3. Recent Advances in Adaptive Middleware: the DARPA Quorum Program

To investigate and transition many of the capabilities required to provide end-to-end QoS to mission-critical distributed and real-time systems, DARPA established the Quorum research program. This DARPA program focuses on Quality of Service (QoS) in the broader meaning of the term, i.e., as an organizing concept for integrated, adaptive resource management for mission-critical distributed computing applications.

The Quorum program is developing the technologies, as well as assembling them into an integrated, distributed infrastructure environment, that are necessary to support the QoS requirements of next-generation mission-critical applications, such as those described in section G. A fundamental premise of Quorum is that adaptive QoS management is a key organizing paradigm for providing applications with the end-to-end assurances necessary to guarantee mission success in highly dynamic, unpredictable, heterogeneous networked environments, both for embedded contexts and the more widely dispersed collaborative use cases. While emerging network-level QoS mechanisms, such as RSVP and Internet2, are an essential enabling technology for Quorum, they are not sufficient *in isolation* because they are limited to QoS at the communication layer. In contrast, Quorum defines "end-to-end" as being the quality-of-service seen by the application, which calls for coordinated QoS management *across* middleware, operating system, network, and other communication layers.

2.4. Quorum Program Concepts

An overview of the Quorum program concept is shown in Figure 2.1. The top half of the figure illustrates the concept for the technology needed to provide adaptive feedback paths that maintain and adjust the quality delivered to applications end-to-end. The "drilling" that penetrates the layers below depicts the need to coordinate the information and control flow between layers according to the end-to-end QoS needs. The lower part of the figure connotes the multiple, early transition targets for these technologies, ranging from a focus on embedded computing to integrated command and control systems.
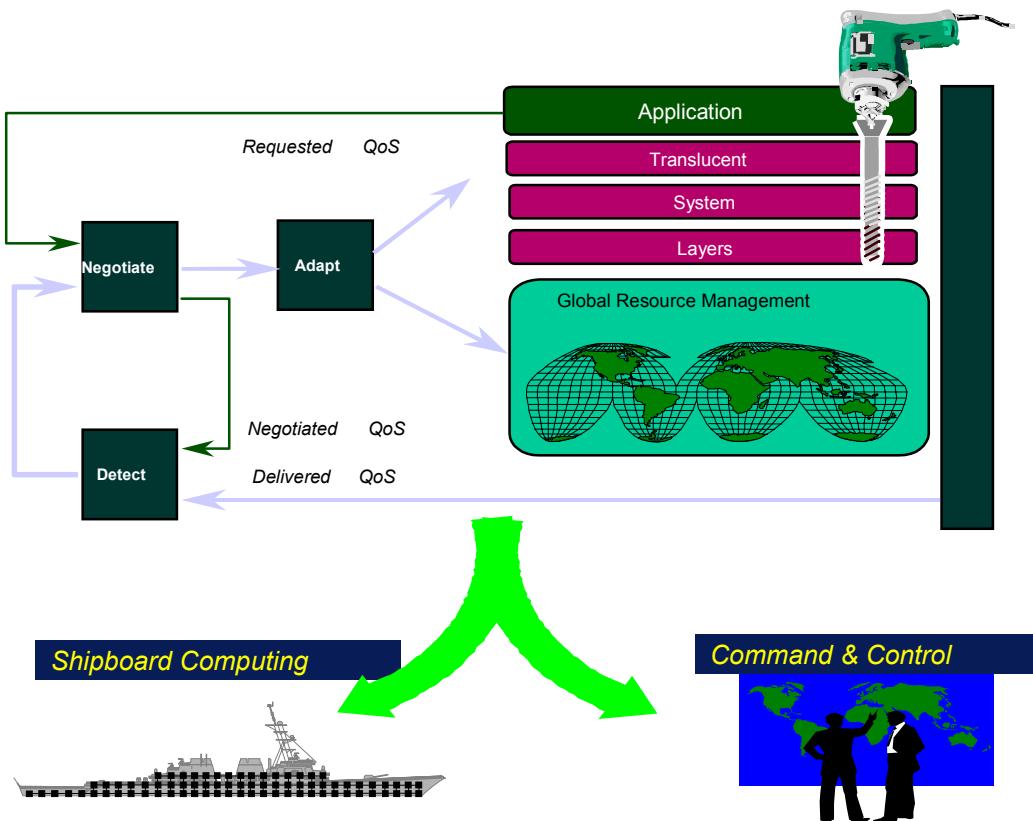
Figure 2.1: Quorum Program Organization

The Quorum program is structured according to the following interrelated technology development tasks and the integration and demonstration task.

Quality-of-Service (QoS) Framework
This task focuses on developing the end-to-end QoS management technologies that define the innovative architectural principles of Quorum. QoS is broadly defined to encompass not only performance measures such as bandwidth, latency and throughput, but also mission-critical aspects such as real-time constraints, dependability and security.

Translucent System Layers
This task focuses on developing the innovative system software layers needed to populate the QoS framework architecture. In contrast to the classical ideal of "layered functional transparency" in distributed computing, in which implementation decisions that impact mission-critical QoS aspects are invisible to the user and frustratingly uncontrollable, this task pursues the concept of *translucency*. Translucent services preserve the benefits of functional transparency, but are dynamically responsive to QoS constraints imposed by higher layers (or feedback from lower layers or the environment). This design principle allows these services to adapt their behavior through selection or specialization of alternative implementations, policies or mechanisms in accordance to changing requirements and environments.

Adaptive Resource Management
This task focuses on developing the resource management strategies necessary to dynamically discover, allocate and schedule resources in accordance with QoS constraints negotiated by applications and the system infrastructure. Associated issues include collection and maintenance of a consistent global view of resource status, as well as dynamic resource allocation algorithms that support adaptation in response to workload demands, failures or crisis mode behavior.

Integration, Demonstration and Validation
Realizing the ambitious Quorum vision requires coordinated development of the constituent technologies outlined in the preceding three bullets, together with their integration into a series of concept demonstration reference implementations of successively greater capability, as well as their evaluation and demonstration in the context of production-quality mission-critical applications. Currently available parts of the Quorum technology base have already been used successfully in a range of applications and demonstrations including avionics mission computing at Boeing [6] and video control for a simulated unmanned aerial vehicle with adaptive real-time behavior [41].

The QuOIN activity operates across all four of these task areas.

2.5.   Organizing Technical Themes

For a significant number of mission critical systems, both the current operating environment and information processing requirements can be expected to change during operation. This often implies the need to adapt to changing conditions. One Quorum challenge is to design a distributed system infrastructure-based architecture and reify this architecture into a concrete reusable middleware framework for building adaptive applications. As shown in Figure 2.2, the Quorum architectural framework is organized around QoS-oriented specification, feedback, negotiation and control

QoS Param 2

QoS Param 1

Region-Based QoS Management

Requested QoS

Application

Negotiate

Adapt

System

Negotiated QoS

Detect

Delivered QoS

Feedback Control Paradigm
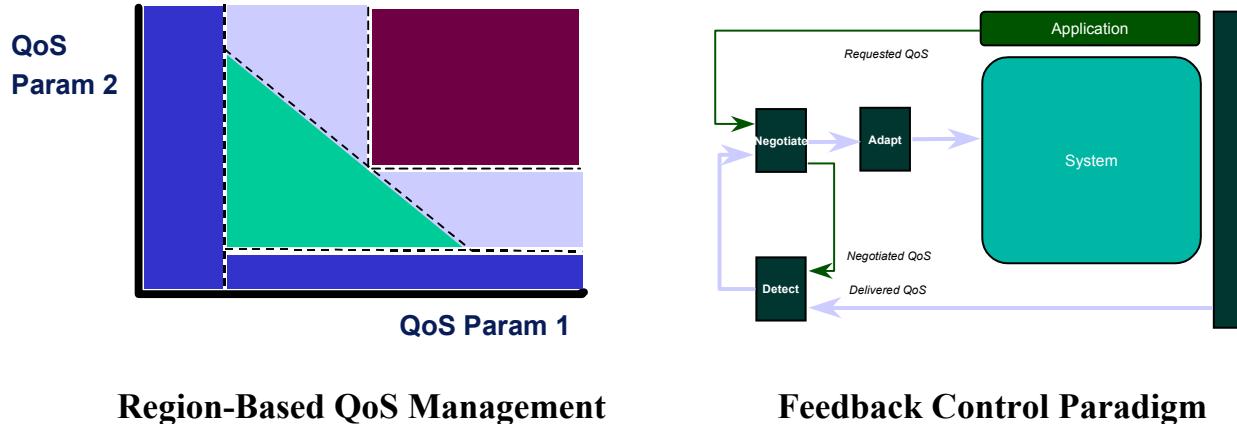
Figure 2.2: Key Quorum Program Technical Challenges

mechanisms that can be applied to the wide range of QoS aspects currently under investigation in Quorum.

Quorum is developing advanced, reusable middleware that can enable a new generation of flexible distributed and embedded applications. These new applications will have more explicit control over their resource management strategies. As a result, they will be more easily reconfigurable and adaptive in response to dynamic changes in network and computing environments. In particular, they will have a wider operating range than the conventional prevailing binary mode between "working" and "broken."

From one perspective, Quorum is developing an extensible software development framework, built on a distributed object (DOC) middleware infrastructure that simplifies the support of dynamic run-time adaptation to changing configurations, requirements or availability of resources. From a complementary perspective, it is an evolving architecture with a growing base of components and mechanisms filling out this architecture to support an integrated QoS concept for managing collections of system aspects and the tradeoffs among these aspects to support varying operating objectives.

2.6.   Technical Integration

In any DOC middleware architecture, the *functional path* is the flow of information between a client's invocation to a remote object and back. The middleware is responsible for exchanging this information efficiently, predictably, scalably and securely between the remote entities by utilizing the capabilities of the underlying network and endsystems. The information itself is largely application-specific and determined solely by the functionality being provided (hence the term "functional path"). The functional path deals with the "what" of the client⇔object interaction from the perspective of the application, e.g., what function is to be requested for that object, what arguments will be provided and what results, if any, will be returned to the client.

In addition to providing middleware that supports the functional path, Quorum adds a *system path* (a.k.a., the "QoS path") that handles issues regarding "how well" the functional interactions behave end-to-end. Thus, Quorum middleware is also intimately concerned with the *non-functional* aspects of

distributed and embedded application development. For example, this involves the resources committed to client ⇔ object interaction and possibly subsequent interactions, proper behavior when ideal resources are not available, the level of security needed, the recovery strategy for detected faults, etc. A significant portion of the Quorum middleware is dedicated to facilitating the collection, organization and dissemination of the information required to manage how well the functional interaction occurs, and to enable the decision making and adaptation needed under changing conditions to support these non-functional "how well" QoS aspects.

Quorum separates the non-functional QoS requirements from the functional requirements for the following two reasons:
- To allow for the possibility that these requirements will change independently e.g., over different resource configurations for the same applications; and
- Based on the expectation that the non-functional aspects will be developed, configured and managed by a different set of specialists than those customarily responsible for programming the functional aspects of an application.

In its most useful forms, the non-functional QoS aspects extend end-to-end. Thus, they have elements applicable to the network substrate, the platform operating systems, the distributed system services and the programming system in which they are developed, the applications themselves, as well as the middleware that integrates all these elements together. Thus, the following two basic premises underly adaptive middleware:
- Different levels of service are possible and desirable under different conditions and costs, and
- The levels of service in one dimension may need to be coordinated with and/or traded off against the levels of service in other dimensions to achieve the intended overall result.

There are three complementary parts to Quorum's middleware organization:
- The first part deals with the features and components needed to introduce the concepts for predictable and adaptable behavior into the application program development environment, including specification of desired levels of QoS aspects.
- The second part deals with providing run-time middleware to ensure appropriate behavior, including collecting information and coordinating any needed changes in behavior.
- The third part deals with inserting the mechanisms for achieving and controlling each particular aspect of QoS that is to be managed, including aggregate allocation, scheduling and control policies.

Integrating these facets and inserting sample mechanisms and behavior is a significant integration job. Therefore, the Quorum Object Integration (QuOIN) activity is combining technology developed or derived from a number of individual Quroum projects to demonstrate both how and how well various layers interact.

Figure 2.3 illustrates our general concept of middleware and some of the key layers we are using to organize our QuOIN technology integration activities. Based on our prototyping and benchmarking activities to date [6, 17], the integrated QuOIN components enable an unprecedented degree of application level control and adaptability to varying conditions typically found in both embedded and Internet environments, as they are occurring. In turn, these integrated capabilities enable a new generation of applications that are designed to be easily customized in their resource management, without the need to make the task of the application developer significantly more complex or risky.
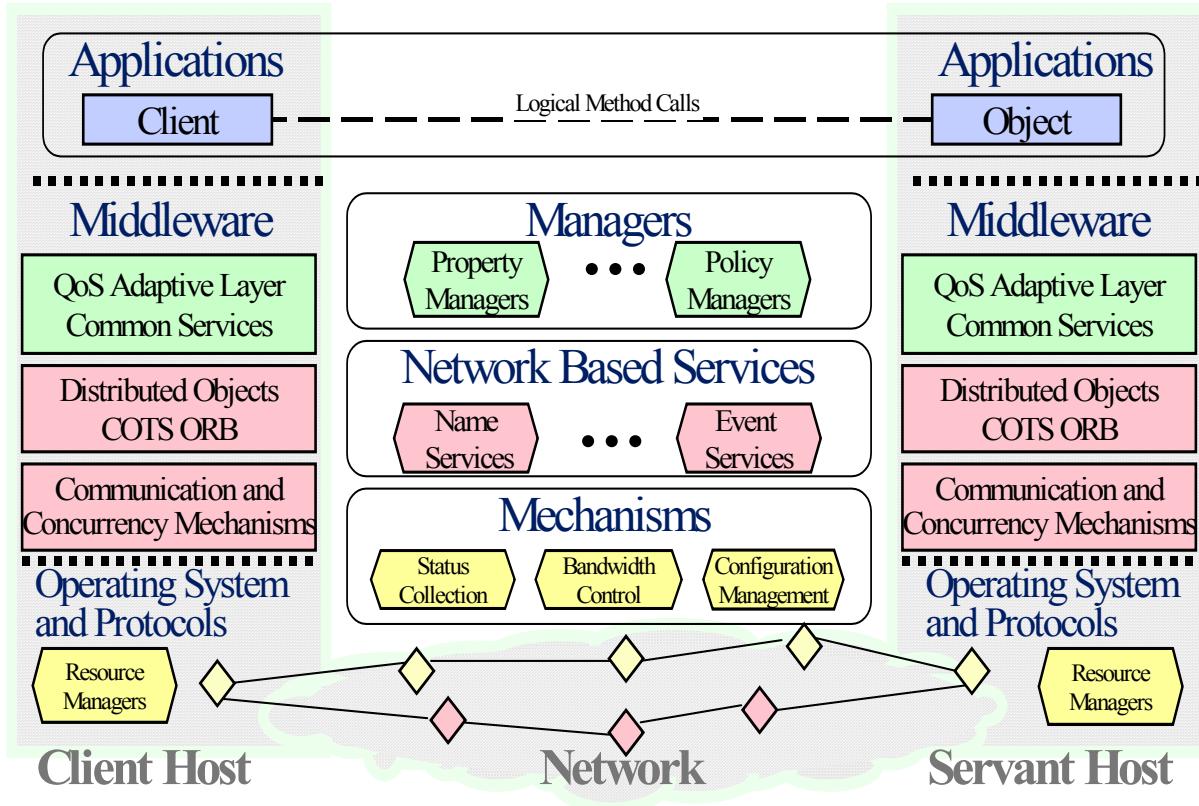
Figure 2.3: Simplified System Model

In the remainder of this section, we highlight the QuOIN project activities we are using to populate the Quorum adaptable system vision in order to complete the necessary infrastructure components and specific QoS property mechanisms. Our intent is to provide a brief overview of the Quorum components being integrated by QuOIN. Each of these activities has been described in more detail elsewhere, and we provide pointers to the individual project web sites for additional information on the particular technologies.

2.6.1. The TAO Real-time ORB

The TAO ORB was developed at Washington University, St. Louis and the University of California, Irvine to provide an open-source implementation for a CORBA-compliant, real-time, COTS middleware ORB and related ORB services. The TAO ORB endsystem contains the network interface, OS, communication protocol and CORBA-compliant middleware components and services illustrated in Figure 2.4.

TAO supports the standard OMG CORBA reference model and Real-time CORBA specification [19], with enhancements to ensure predictable QoS behavior for real-time applications. In particular, TAO provides a real-time object adapter, run-time schedulers for both static and dynamic real-time
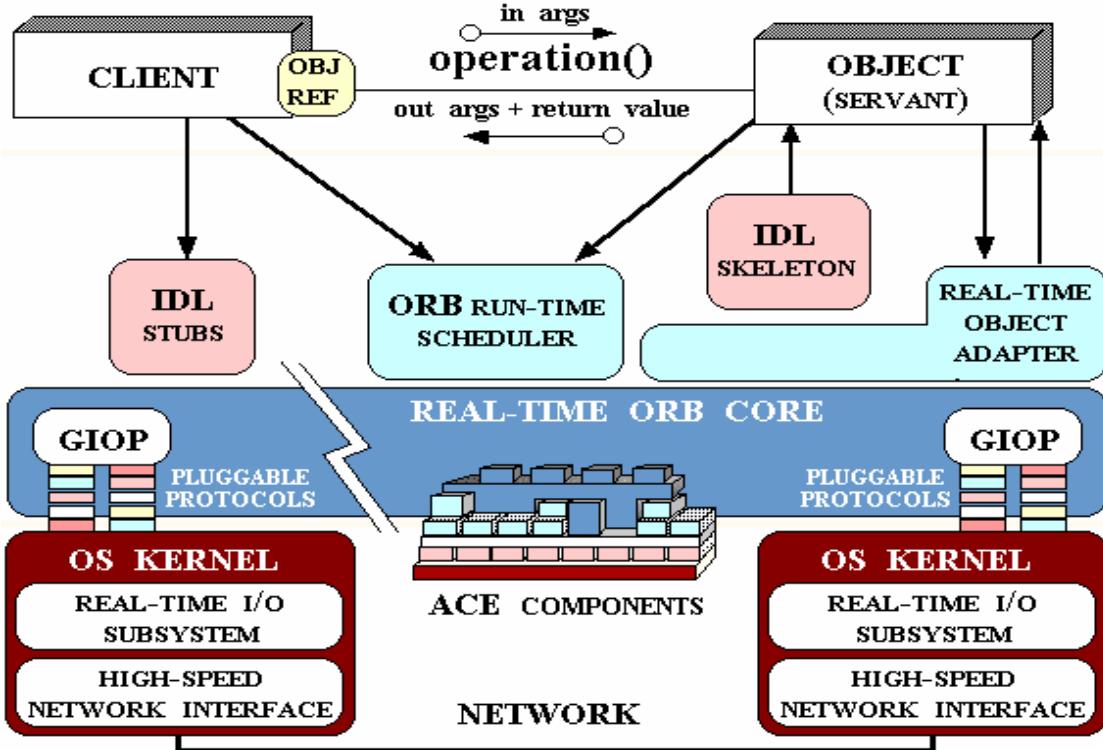
Figure 2.4: Real-time CORBA ORB and Related Subsystems

scheduling strategies [30], and a real-time event service [10], which enables applications to utilize a publish/subscribe pattern of real-time development within a CORBA context.

2.6.2. Quality Objects (QuO)

The Quality Objects (QuO) [40, 35, 15] CORBA-based middleware framework was developed at BBN to facilitate the creation and integration of distributed and embedded applications that can specify (1) their QoS requirements, (2) the system elements that must be monitored and controlled to measure and provide QoS and (3) the behavior for adapting to QoS variations that occur at run-time. QuO adds the abstractions of a *QoS contract* that summarizes the service requirements of the possible states the system might be in and actions to take when changes occur, *system condition objects* that measure and control behavior, and *delegates* that are packaged adaptive behaviors. In addition, QuO introduces the concept of an *Object Gateway [26]*, which is a means for integrating a wide variety of transport level QoS mechanisms into the DOC middleware paradigm. Figure 2.5 highlights the components of the QuO framework and their relationship to other parts of the Quorum adaptive QoS environment, such as the TAO real-time ORB.

Figure 2.6 highlights the layering of adaptive QoS middleware over the integrated QuO and TAO real-time CORBA DOC environment. In this view, the task of interfacing to the application is assigned to the adaptive middleware layer, which tracks the progress and changes the control parameters accordingly for the enforcement mechanisms provided by the real-time DOC middleware layer. The
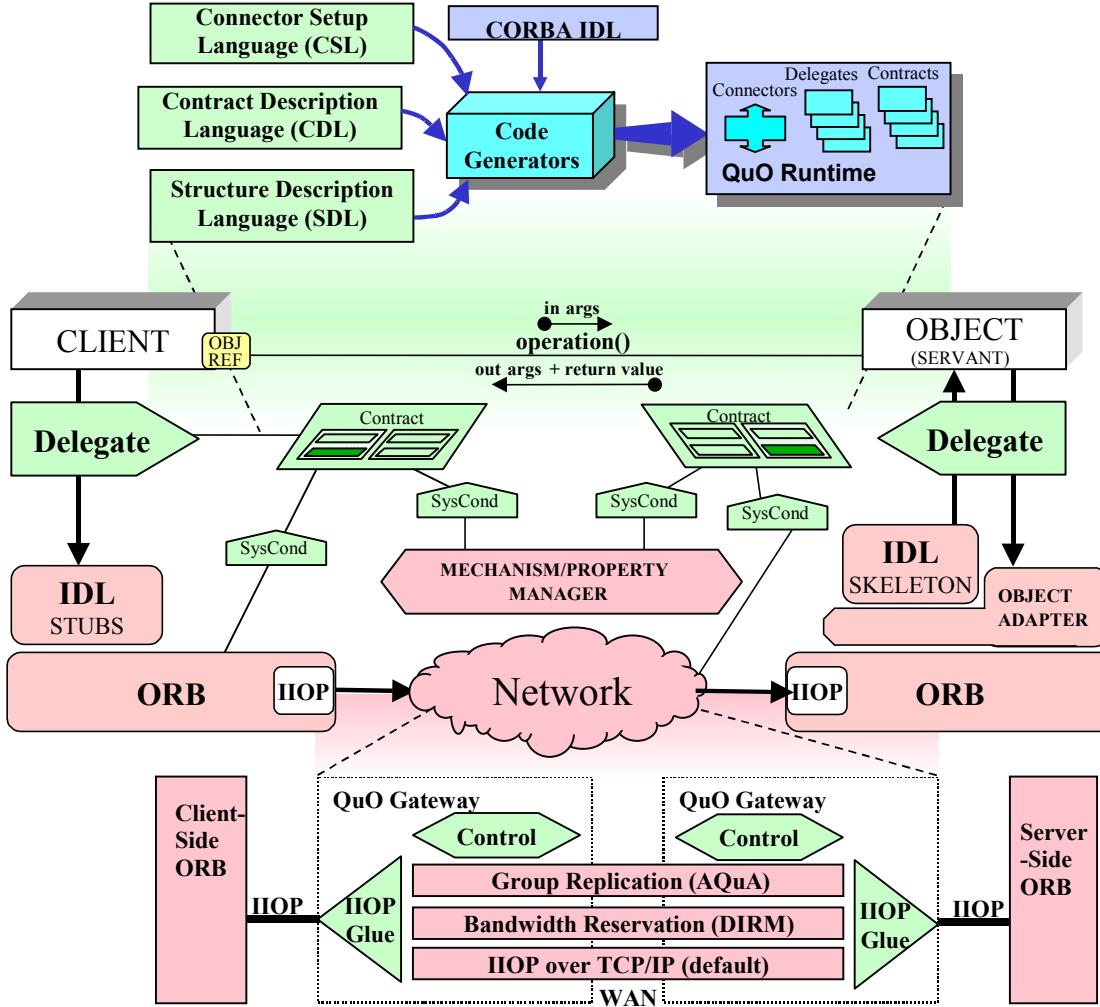
Figure 2.5: QuO Adaptive QoS Framework Components

adaptive QoS management functions as a higher level of middleware that can interoperate with both the application-specific requirements and the lower-level middleware control mechanisms to produce the desired behavior under varying circumstances. Real-time CORBA mechanisms, such as prioritization and filtering, can be modified dynamically to better match the current application requirements with the current operating conditions, as measured and evaluated by the adaptive layer.

2.6.3. Proteus Dependability Manager

Figure 2.7 highlights the design for the dependability aspects in QuOIN. Developed primarily at the University of Illinois, this design is based on using off-the-shelf group communication mechanisms (Ensemble) to control the consistency of object replicas. It provides a prototype property (dependability) manager component, named Proteus, which coordinates the number and location of object replicas, as well as coordinating the selection of replica control strategy, from among a growing class of supported strategies with various footprint and fault coverage capabilities. For more details on the design for dependability see [3, 25] and [18] for the emerging CORBA Fault Tolerance standard which this work has influenced.

13
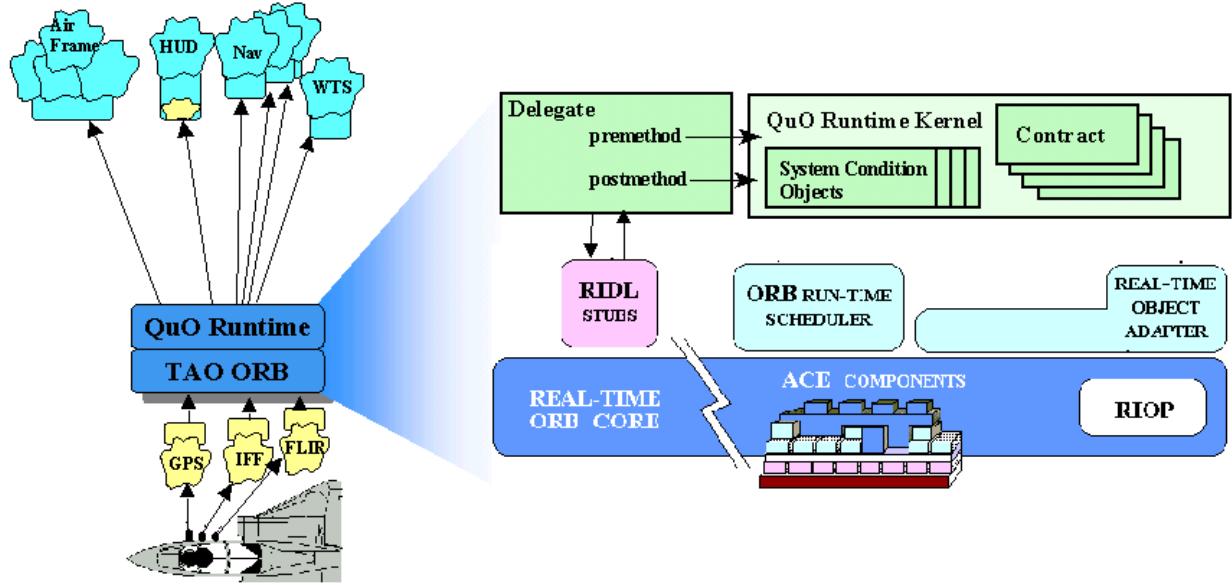
Figure 2.6: Adaptive Real-Time Behavior

2.6.4. Remos and Darwin Network Measurement and Control Functions

Measurement and control of network resources form an important part of the bandwidth management QoS capability. Two components from Carnegie Mellon University, REMOS and DARWIN, are being developed in Quorum to provide advanced bandwidth management capabilities. REMOS is a subsystem for acquiring and disseminating information about network (and host) resource utilization, and an emerging capability for predicting future use. DARWIN [23] is a subsystem for hierarchically managing network resources, and provides a resource reservation mechanism capability to the QuOIN integration. Figure 2.8 illustrates how these two technologies integrate with other components in the Quorum DOC framework.

2.6.5. Object-Oriented Domain Type Enforcement Access Control

Adaptive security is supported in the QuOIN environment through another Quorum technology called Sigma, being developed by Network Associates/TIS Labs. Sigma is a DOC-based access control mechanism and policy language that employs a domain type enforcement model. Introducing Sigma into the QuOIN environment involves providing adaptive security policies, enforceable through the ORB and the Object gateway. In addition, it provides a response mechanism that can be connected to a variety of triggers, such as variations in delivered QoS or specific Intrusion Detection Systems (IDS), to form the basis of defensive actions taken under suspicious circumstances. Figure 2.9 illustrates the concept integrated with DOC. For more details on the adaptive security aspects of this work, see [32, 17, 36].

There is a substantial amount of information about QuOIN and the QoS mechanisms developed by our research partners available via the Web. Below we list a sampling of pertinent URLs.
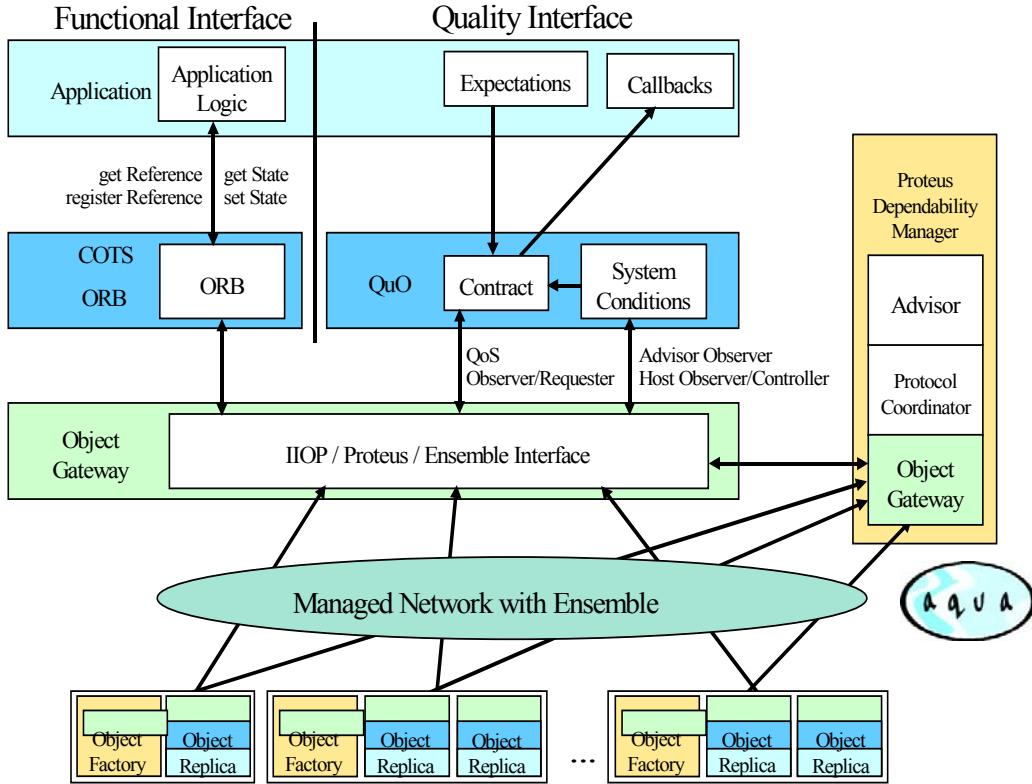
Figure 2.7: Dependability Management Using Group Communication

For information on the Quality Objects (QuO) adaptive QoS middleware framework, visit BBN's QuO site at http://www.dist-systems.bbn.com/tech/QuO/.

Copies of many of the QuO technical publications are available at http://www.dist-systems.bbn.com/papers/.

TAO (The ACE ORB) is the high-performance, real-time middleware underlying much of the Quorum CORBA infrastructure; it provides QuOIN's real-time aspects. For information on ACE and TAO (The ACE ORB), visit Washington University's web site at http://www.cs.wustl.edu/~schmidt/TAO.html.

The AQuA project, a joint effort of BBN and University of Illinois, with support from Cornell University, has developed QuOIN's dependability module. For additional information on AQuA, visit the AQuA web site at http://www.crhc.uiuc.edu/PERFORM/AquA.html.

The RSVP managed bandwidth capability grows out of joint research by BBN and Columbia University conducted under the DIRM project. Additional information on this research is available at BBN's DIRM site at http://www.dist-systems.bbn.com/projects/DIRM/.
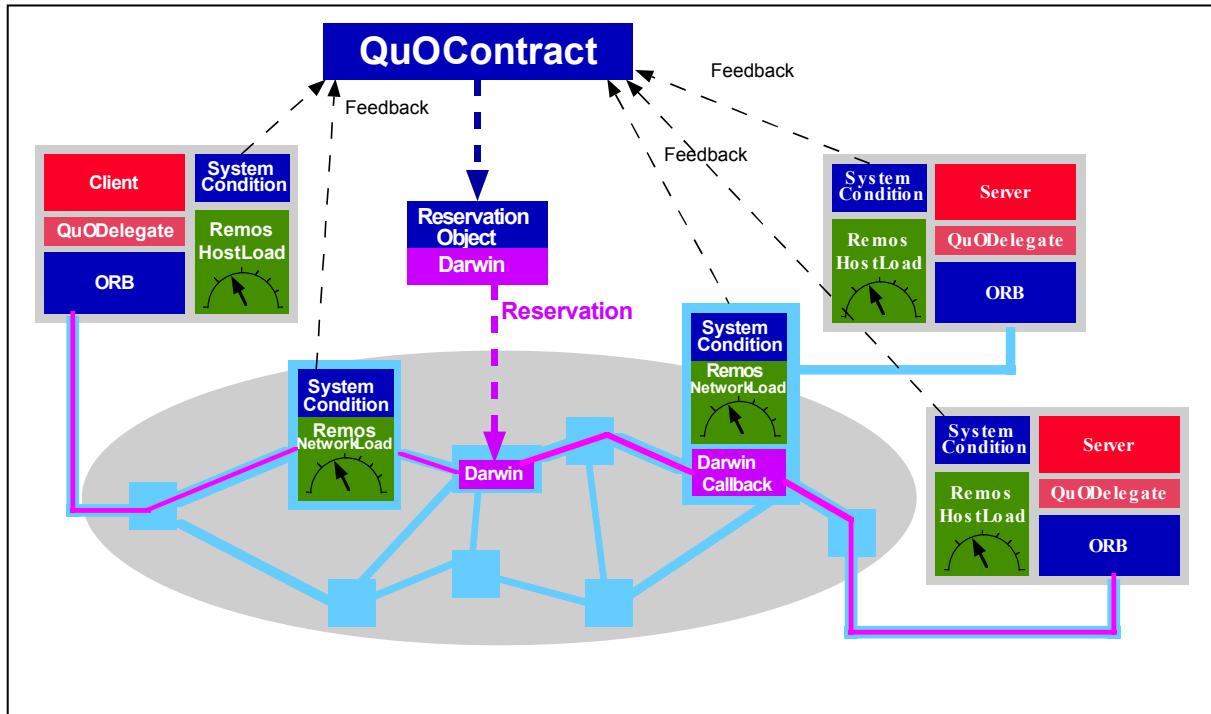
Figure 2.8: Network Status and Control

The QOSockets capability that underlies the RSVP managed bandwidth property is described in more detail at Columbia's QOSockets site at http://www.cs.columbia.edu/dcc/qosockets/.
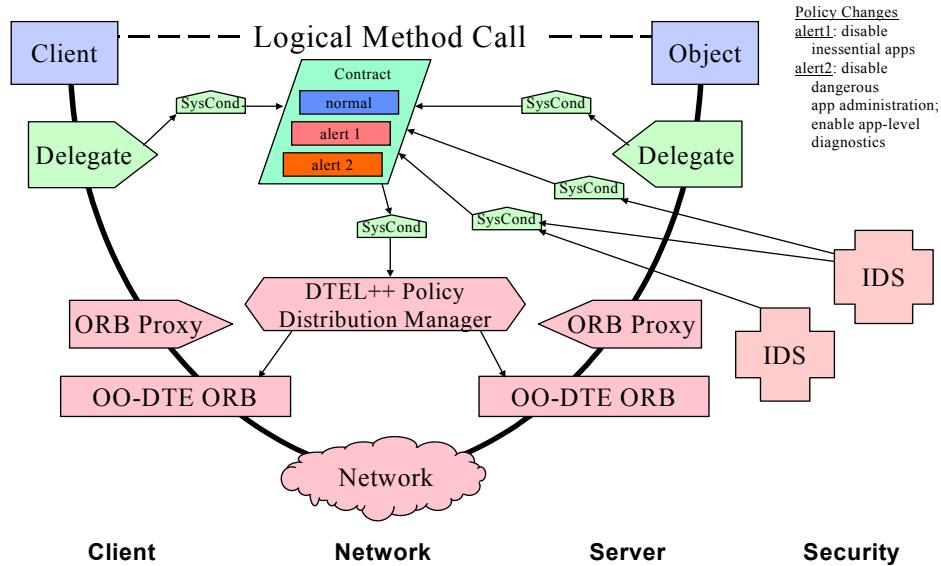
Figure 2.9: Adaptable Access Control Policy Components

For additional information on CMU's Remos project, which supports QuOIN's status monitoring capability, visit CMU's Remos site at http://www.cs.cmu.edu/~cmcl/remulac/remos.html.

16

Some of our work in the managed bandwidth area is based on CMU's Darwin project. For more information visit http://www.cs.cmu.edu/~darwin/.

For additional information on Network Associates/TIS Labs' SIGMA research, which forms the initial basis of the QuOIN security / access control capability, visit http://www.nai.com/products/security/tis_research/applied/arse_corba.asp.

The Quorum technologies selected for integration were chosen carefully because they focus on complementary parts of the end-to-end solution for different QoS aspects, and because they had already self-selected toward common COTS middleware implementation base. However, integration at the level of large-scale technology investigation projects is very hard. Part of the difficulty stems from the simultaneous investigation and development of both the common integrating environment, as well as the individual technology focus. Since these are both leading-edge activities, key new developments from each often must be factored into the others. Therefore, one of the key lessons we've learned so far is that it has been very effective to have an overall middleware integration framework that can incorporate the key ideas from a variety of QoS dimensions. However, the complementary lesson learned is that integration boundaries must be fluid and are generally nominal, pending a multi-way negotiation to mediate between the individual technology needs and the need for common integration. We have found that CORBA provides an effective middleware integration framework to suit many of the needs of QuOIN.

It is (and has been) significantly less effective to attempt to integrate blackbox solutions. Rather, in the most effective cases, the integration itself has been the focus for development activities to accommodate changes in the QoS mechanisms to account for integration, and changes in the common environment to account for the individual needs of the QoS mechanism. The degree of change can be expected to diminish with experience. However, in our opinion, there will be constantly evolving boundaries as insights and innovation occur across the individual mechanisms and the integrating medium.

2.7. Examples of Distributed, Realtime Embedded Applications Using Adaptive Middleware for Demonstration and Technology Transition

This section discusses two example DRE applications we've built, evaluated and experimented with as part of our investigation of managing QoS and adaptive behavior to meet mission requirements under varying and changing operating conditions, while simultaneously testing and evaluating emerging Quorum technologies. The first of these applications (WSOA) is part of a fielded concept of operation in conjunction with the Boeing Company and AFRL supporting dynamic inflight replanning. The second (UAV) represents a real-time video sensor capture, dissemination and processing capability with feedback, linking mobile and fixed assets, operating over shared resources.

2.7.1. Weapons System Open Architecture (WSOA)

The Weapons System Open Architecture (WSOA) [42] program has explored the possibility of building an open-systems testbed environment in which legacy embedded systems in the avionics
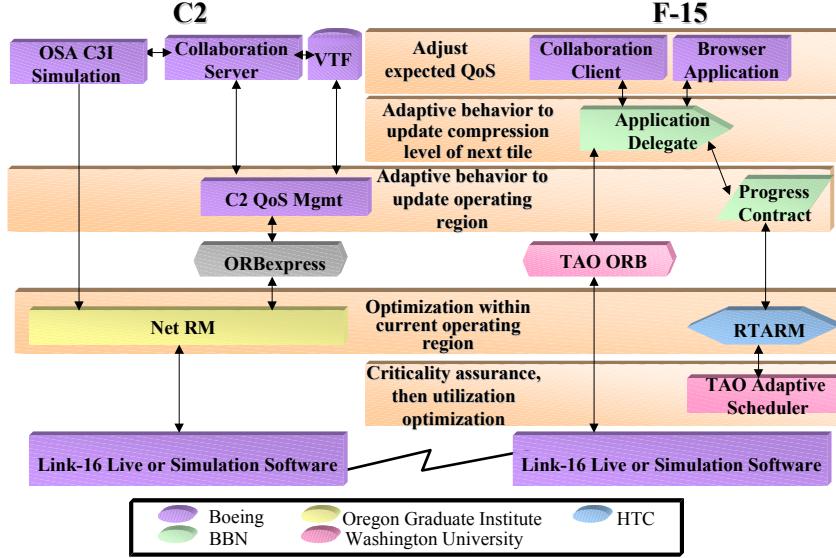
Figure 2.10: WSOA dynamic replanning system architecture

domain can be integrated within a next-generation middleware quality of service (QoS) management context to provide unprecedented capabilities for time-critical target prosecution. The major feature areas provided by the WSOA framework (Figure 2.10) are as follows:

- Tactical communication links, using legacy Link 16 technology to connect mission computers on an F-15 aircraft with imagery terminal servers on a command-and-control (C2) aircraft.
- Collaborative planning between F-15 and C2 personnel during a mission.
- Giving F-15 personnel information-mining capabilities on the C2 imagery libraries, via a browser-like interface using standard F-15 cockpit equipment.
- Link bandwidth optimization, using contract-based evaluation of application requirements and system resources and loads.
- Adaptive and dynamic QoS management, to ensure maintenance of critical assurances, while optimizing and tuning non-critical performance.

This discussion highlights the last two feature areas: link bandwidth optimization and adaptive and dynamic QoS management. To provide assurances and optimization of key QoS system properties, coordinating diverse approaches to a number of QoS management areas is fundamental. In particular, we have integrated several forms of advanced middleware capabilities within the WSOA testbed, including the following technologies that constitute layered middleware architecture (Figure 2.11):
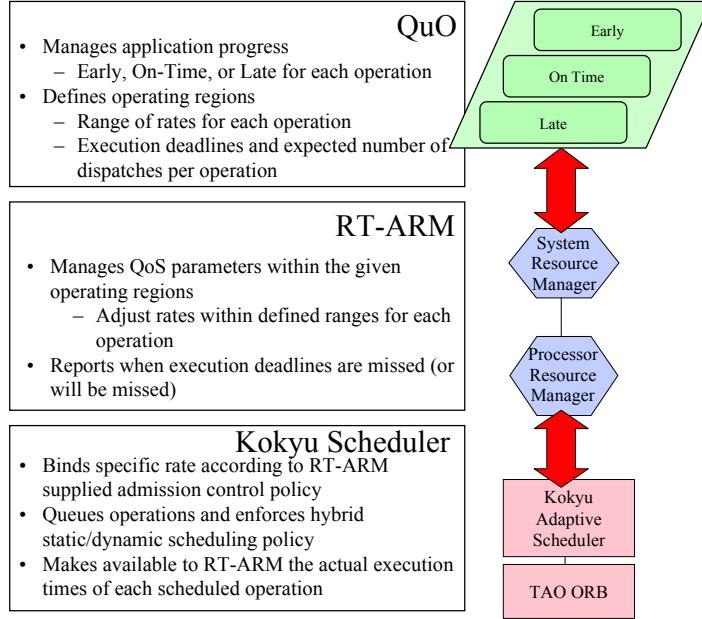
Figure 2.11: Layered resource management

- QuO, an end-to-end QoS management middleware framework developed at BBN Technologies,
- RT-ARM [45], a real-time adaptive resource manager developed at Honeywell Technologies,
- Kokyu [46], a real-time scheduling and dispatching framework developed at Washington University in St. Louis,
- TAO, a high-performance and real-time CORBA-compliant object request broker (ORB) developed at Washington University in St. Louis and the University of California, Irvine.

In this subsection we consider how the WSOA project provides insights into the nature of resource management, dependability and adaptivity in heterogeneous systems with demanding QoS requirements. Of particular interest is how the balance of (1) strict assurances for critical system behavior and (2) adaptive tuning and optimization of non-critical behavior is achieved. Each middleware layer addresses separate concerns, and yet those concerns must be woven seamlessly end-to-end and layer-to-layer to achieve robust and dependable system performance in complex mission-critical systems, such as the WSOA testbed framework.

2.7.1.1. *Robust behavior during overload*

One key area in which dependability must be considered is whether skillful management of resources can allow a more robust response to scarcity in worst-case conditions, as well as giving better overall performance in the average case. In WSOA, real-world constraints on power consumption, weight and frequency of processor upgrades limit the ability to offer sufficient resources through excessive over-provisioning. For example, the complexity of the environment in which WSOA is expected to operate combined with the further complexity of systems-of-systems integration end-to-end, means that explicit testing of all possibilities is not possible. Instead, other techniques, such as combining testing and model-based analysis must be pursued.

While the *total* system requirements for the WSOA testbed are both high in quantity and variable in the face of environmental variations, a key insight is that systems of this kind may still be engineered

so that a *critical subset* of the total requirements can be implemented so that they are both (1) lower in quantity and (2) more predictable in both time and in interaction with other system features. For example, operations can be designed in a way similar to the imprecise computations model in that a fixed *sufficient* number of computation steps could be performed by a single critical operation, with additional *desirable* computation steps being performed by a sequence of non-critical operations whose execution is not assured. This means that resources must be provisioned to assure the maximum *critical* requirements can be met, and that additional processing be managed dynamically and adaptively with remaining resources.

2.7.1.2. *The changing nature of proper functioning*

In an avionics mission-computing environment, such as WSOA, operations may exhibit different degrees of criticality, depending on the environmental and system context. For example, consider computing the next segment of a navigation route, with each segment going between two waypoints. In some situations, such as under low-threat conditions, the first two segments from the aircraft's current position may be the only ones considered critical, and others can be designated non-critical to reduce the demands of the critical subset of processing. In other situations, such as egress from a high-threat environment, more route planning may be considered critical, so that additional navigation operations must be added to – and presumably other kinds of processing can be removed from – the critical subset.

Interestingly, non-critical processing may also be made more dependable through context-aware adaptive management. For example, if a non-critical navigation route segment cannot be computed on time, the application might retry the computation rather than proceed to compute the next segment, whose origin would depend on the one that was not computed. As a similar example, imagery is downloaded as a sequence of tiles of varying quality, radially from a point of interest in the image. WSOA uses the QuO middleware to monitor download progress, and tune image tile compression upward as necessary to fit within the specified image transmission time (Figure 2.12). Clearly, image tiles near the point of interest must be kept at as low compression as possible to improve image quality – however, additional surrounding context in the image may still be of sufficient resolution at lower quality to be useful.
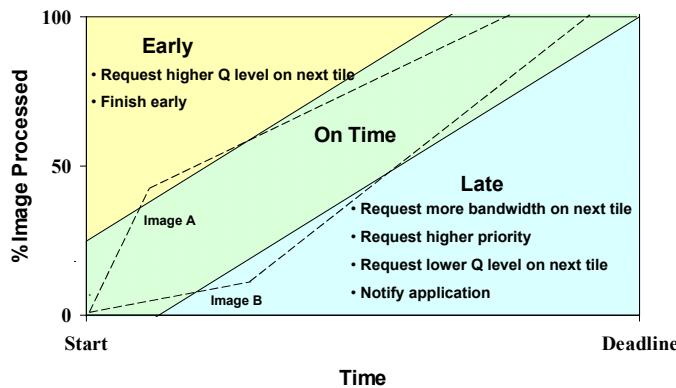
Figure 2.12: A QuO contract monitors the progress of image download and adapts to keep it within the "On Time" range

The issue of timeliness for imagery download is interesting in the WSOA context due to the complexity of the libraries themselves, and the degree to which compression levels depend on the contents of the images. While a QuO contract may request a level of compression matching expectations of timely download and sufficient quality, the image itself may result in discrete alternative compression levels for each tile. Moreover, system load, user inputs and other factors may further perturb the conditions under which QoS is managed.

A control-centric QoS management architecture is therefore necessary to maintain dependable system behavior in the face of unanticipated conditions. QuO monitors and adjusts its perception of system behavior regularly, so that it maintains a clear picture of the *actual* conditions under which it is controlling system QoS. RTARM monitors a different set of conditions, such as whether it is succeeding in meeting control boundaries for processing (decompression, storage and delivery) tiles that arrive at the F-15 endsystem, and when it cannot feasibly schedule that processing either directs Kokyu to reschedule the CPU with some operations at lower ranges of available rates, or reports back to QuO that it cannot meet its obligations and QuO can respond accordingly. What is most important here is that the control loop is *closed* even though it crosses multiple QoS management layers.

2.7.1.3. Moving design and resource management decisions to later in the software engineering cycle

In the Bold Stroke domain-specific middleware, upon which the WSOA framework is based, late binding was applied initially to software, in an evolution from handcrafted assembly language, to an object-oriented component-based CORBA approach written in C++. The effect of this evolution was that the point in the software lifecycle at which functionality could be applied was pushed significantly later. For example, in the handcrafted version, all functionality had to be defined before the design of a cyclic executive that ran the components. In the initial CORBA version, rate monotonic scheduling was done at system build time, and through careful design of system modes, the set of objects that was active could be selected at run-time.

WSOA takes this idea of late binding to another level, particularly in the area of QoS management. Hybrid static/dynamic scheduling and adaptive selection of execution rates was applied by Kokyu to defer operation scheduling until run-time. The RTARM performed adaptive monitoring and adjustment of available rates to maintain system operating conditions within specified limits. QuO used contract-based monitoring and evaluation to integrate diverse resource management techniques (*e.g.,* image compression levels and rates of execution for the decompression operation) and to ensure that QoS requirements were met end-to-end.

2.7.1.4. Knitting the different perspectives into a unified management capability

The various capabilities described above need to be integrated together to form a cohesive system-level solution, which involves information sharing across and between these views. Each of the three levels of adaptive QoS management described above, QuO, RTARM and Kokyu, operates on a different time scale. QuO can make image compression adjustments on a per-tile basis, with a maximum rate on the order of seconds. RTARM also bases its decisions on rate selection, and its maximum responsiveness is therefore also on the order of seconds. Note that QuO might make both a compression level decision and handle a report of failure to maintain the given QoS limits from RTARM in the same interval, though such case are expected to be infrequent. The Kokyu scheduler makes operations scheduling decisions whenever requested by RTARM, so its rate is naturally tied to

the RTARM. As noted below, however, Kokyu is designed to minimize the fraction of that interval that it uses, to minimize overhead of adaptive responsiveness and improve the *possible* responsiveness, *e.g.*, if the RTARM were calibrated to run at a faster rate. Finally, the Kokyu dispatcher makes dynamic scheduling decisions and multiplexes dispatch requests onto prioritized threads, on the order of ten microseconds. As a result, as little overhead is added to the in-band (as compared to Kokyu scheduling, RTARM and QuO adaptation, which operate out-of-band) path as possible.
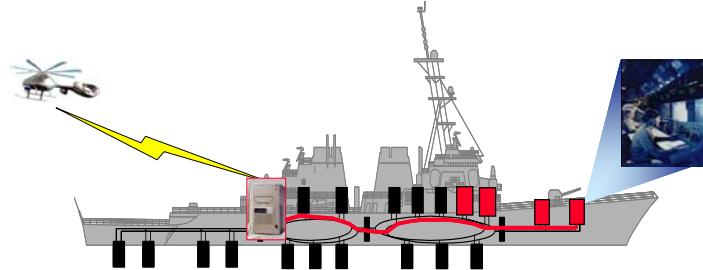
Figure 2.13: The UAV concept includes information dissemination from, and control of, UAVs

2.7.1.5.   *Experimentation*

To measure the benefits of multi-layer adaptive management, we are in the process of conducting experiments within a realistic F-15 OFP hardware and software environment. We have instrumented the adaptation paths through the application to collect time stamps around the following activities:

- QuO contract evaluation
- QuO-triggered adaptation of compression levels
- QuO-triggered adaptation of available rates of execution for image tile processing operations
- RTARM in-region evaluation
- RTARM-triggered adaptation of available rates for image tile processing operations and
- Kokyu scheduler adaptive rescheduling of operations.

In addition, we have instrumented the image tile request-download-decompression-delivery path to assess the impact of adaptive management on application performance. We will run three major experiments, with repeated trials of each: (a) full compression, without adaptation, (b) no compression, without adaptation, and (c) variable compression, using adaptation.

By obtaining and analyzing these data, we will achieve a clear profile of the following key factors for multi-layer adaptive resource management:

- Coupling of layers in time and in overhead
- Interactions of adaptation at different time-scales
- Overhead measures for adaptation at each layer
- Impact of adaptation on application performance – particularly how adaptation provides acceptable quality and timeliness of imagery, compared to over-compressed or under-compressed approaches without adaptation.

2.7.2. The Unmanned Aerial Vehicle (UAV) Application

A second example we've developed is an unmanned aerial vehicle (UAV) [43] demonstration. As part of an activity for the US Navy, DARPA and the US Air Force, we have been developing a DRE system centered on the concept of information dissemination from, and control of, UAVs, as illustrated in Figure 2.13. In this application we are concerned with managing the QoS requirements for (a) the delivery of video from UAVs to a command and control (C2) platform (e.g., a shipboard environment, ground station, or air C2 node) and (b) the delivery of control signals from the C2 platform back to the UAVs. This QoS management includes trading off video quality and timeliness, and coordinating resource usage from end-to-end and among competing UAVs to satisfy changing mission requirements under dynamic, and potentially hostile, environmental conditions.

Figure 2.14 illustrates the architecture of the demonstration. It is a three-stage pipeline, with simulated UAVs or live UAV surrogates (such as airships with mounted cameras) sending video to processes (*distributors)* that distribute the video to the proper control stations. The UAVs in our prototypes are implemented by two different types of processes. The first reads MPEG video from a file simulating high frame rate video capture devices. The second is capturing video from a live camera feed, which adds realism to the prototype and provides the ability to manipulate the raw video, but produces frame rates limited by the camera's capabilities. Our prototype also uses both wired and wireless Ethernet connections to simulate the data links from the UAVs to the distributor host. We use the TAO A/V Streaming Service, an implementation of the CORBA Audio/Video Streams standard, to manage the transmission of video between hosts. The wireless links from the second and third UAV surrogates contend for the same wireless Ethernet connection and provide a forum for experimenting with wireless video adaptation strategies. The wired Ethernet connection provides a higher bandwidth connection simulating current and emerging higher-capacity wireless transports.

The video distributor processes send the video streams to control stations on a land- or ship-based network. The control stations include video display processes and other video processing applications (the current demonstration includes an automatic target recognition – ATR – application), each with their own mission requirements.
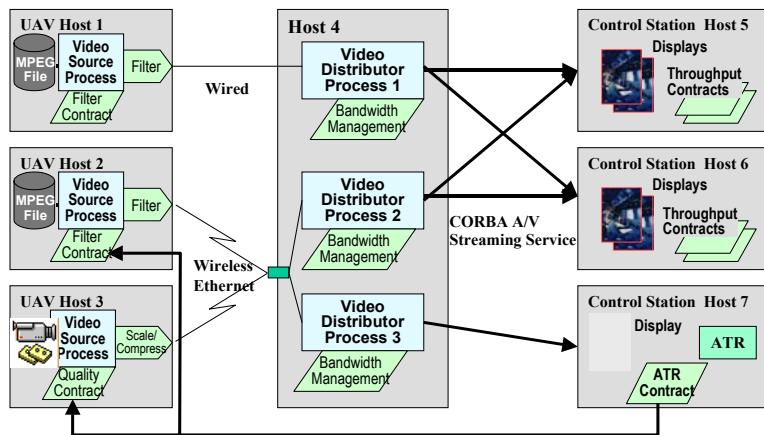
Figure 2.14: Architecture of the Current UAV Demonstration

2.7.2.1.   UAV adaptivity strategies

Adaptation is used as part of an overall system concept to provide *load-invariant performance*. The video displays throughout the ship must present the current images observed by the UAV with acceptable fidelity, regardless of the network and host load, in order for the control station operators to achieve their missions, e.g., flying the UAV or tracking a target. To accomplish this, QuO system condition objects monitor the frame rates and the host load on the distributor and video display hosts. As the frame rate declines and/or the host loads exceeds a threshold, they cause region transitions, which trigger the following adaptations:

- If the network is the constrained resource, send a reduced amount of data, for example by dropping frames or compressing the video at the sender or the distributor.
- If the land-based network is the constrained resource, use a bandwidth reservation protocol to ensure that the distributor is able to send the necessary data to the viewers through the network, even when the network is congested.
- If the distributor CPU is the constrained resource, move the distributor to a different host where more resources are available.

In addition, the ATR and other video processing applications must receive a sufficient frame rate and amount of critical data content to fulfill their missions, e.g., to recognize threats and targets in the video images. To accomplish this, QuO contracts on the UAV video sources coordinate to share the wireless link. For example, the current prototype scales and compresses the video destined for the ATR process while it filters those destined for human display as described above. This maintains a high rate of frames with sufficient information (but not necessarily enough for viewing) for the ATR and a smooth, viewable video for the human operators.

Adaptation is also used to respond to changing mission requirements and to dynamic conditions. For example, in the current UAV prototype, when the ATR process recognizes a target, it changes the mission requirements of the corresponding UAVs. Whereas previously they only needed to make sure a high rate of the critical data needed by the ATR made it through, once a target has been detected the UAV must provide human viewable video so that a commander can make a decision, an operator can track the target, etc. To accomplish this, a contract associated with the ATR host reacts to target recognition by propagating that information upstream to contracts on the UAV sources. The contracts on the UAV coordinate to achieve the new mission, i.e., providing high quality video at a high rate from the targeting UAV in the following manners:

- The targeting UAV shuts off its scaling and compression of the video.
- To accommodate the greater amount of data on the wireless link from the UAV that spotted the target, the other UAVs will reduce their frame rate to the minimal acceptable rate.
- The current prototype includes diversity in the video format (MPEG and PPM), network transport (wireless, LAN, and WAN), and mission requirements (video viewing and ATR) to support a number of experiments in dynamic adaptation.
- The TAO A/V Streaming Service provides the flow connection between the various processes. This is an implementation of the CORBA A/V Streaming Service [47], which supports multimedia applications.
- The CORBA A/V Streaming Service uses RSVP as its bandwidth reservation mechanism.

2.7.2.2.   Robust behavior under overload

Similarly to the WSOA application, the UAV prototype has requirements for dependability even in the presence of load or variations in the availability of resources. Also similarly to WSOA, these include both network resources and CPU resources. The C2 components (whether shipboard or ground-based) that are the target for the UAV video data do not typically face the same constraints on power consumption and weight as the WSOA avionics platform, although the UAVs themselves will frequently face even more strict constraints in these areas. Moreover, the current instantiation of the UAV prototype requires resource management in two dimensions:

- Horizontal, or *end-to-end*, to achieve requirements of video delivery from any particular UAV to the control stations that use it (and control signals from the control stations back to the UAVs).
- Vertical, or *coordinated*, to mediate the conflicting requirements of multiple UAVs, multiple distributors and multiple control stations in any stage of the pipeline, competing for a set of resources, e.g., CPU, network and data.

2.7.2.3.   The changing nature of proper functioning

In the UAV platform, mission requirements can change from control station to control station and from moment to moment, based upon environmental conditions. In normal operation, certain control station functions, such as piloting the UAV, may take priority over other functions, such as non-critical observation. The presence of threats or targets, however, might make the priorities of functions change rapidly. Context-aware adaptation can trade off less important functionality to maintain the requirements of important functionality. For example, we might choose to scale back the video frame rate from a UAV to minimize the latency of the video for a control station that is piloting the UAV. A similar adaptation can be used to maximize the data content of the particular video frames for a targeting officer who must have a high-fidelity image to make command decisions.

The conflicting demands of functional requirements can limit the types of adaptations that are possible and the places at which they occur. For example, if the piloting control station above triggers frame filtering at a UAV source to achieve lower latency, it can affect other stations that are counting on high data content, such as a collector of surveillance data for off-line analysis. A better strategy might be to reserve network resources so that both stations can achieve their requirements: *low latency and high data content*. While some adaptation decisions can be made based only on local information, mission-wide strategies must constrain the adaptation choices available. This is why it is crucial for these adaptation strategies to be programmed in middleware, where they can consider both application-level requirements and system-level mechanisms.

2.7.2.4.   Separation of concerns and late binding of adaptation strategies

The UAV prototype application was developed using the QuO middleware to separate the functional concerns of the application from the adaptation, QoS and dependability concerns. This enabled the functional behavior of the application – video capture, distribution, display and control – to be developed without entangling information about the platform in which the application will be hosted. The development of the UAV "system" then becomes a "construction" process, combining functional components and QoS components into an overall system suitable for the functional requirements, dependability requirements and the characteristics of the target platform once it is known. For example, network resource reservation (which is a reusable component in our UAV prototype) is suitable only if the application requirements can make use of it (i.e., the application needs high bandwidth, low latency) and the platform can support it (i.e., the network supports bandwidth reservation).

This advanced separation of concerns enables us to create an application that is sufficiently flexible and efficient for the requirements and environment for which it is targeted. The alternative, i.e., to encode the adaptation in the functional code of the application, would result in either a tremendously complex and inefficient application that covered every possible contingency in a large set of environments or a more efficient, but brittle application that is dependable only in a small set of environments.

2.7.2.5. Using middleware to knit together a unified capability

The current UAV prototype pulls together QuO adaptation, along with adaptation strategies written in the QuO toolkit, and resource reservations using RSVP. In one deployment of the UAV prototype, i.e., in the NSWC HiPer-D testbed, we integrated with a global resource manager capability. We have also begun exploring using Real-time CORBA, which is a first step towards incorporating an integrated CPU/network resource management capability as part of a complete end-to-end and scalable solution.

2.7.2.6. Experimental Results

We have installed two versions of the UAV prototype application in the Hiper-D lab at the Naval Surface Warfare Center (NSWC) and have evaluated them in the Hiper-D 2000 and 2001 series of integrated advanced concept demonstrations for the Naval Surface Ship domain. These versions have been described in earlier papers [43, 44]. The current version described in this paper is being provided as an open-source release to researchers investigating related issues. The Hiper-D 2000 version used TCP and an earlier, less functional MPEG viewer, and illustrated the use of frame dropping and migration of the distributor in response to excessive processor loads. The Hiper-D 2001 version uses the CORBA A/V Streaming Service, UDP and DVDView, and combines bandwidth reservation, frame dropping and load balancing. The current version is the one described in this paper and adds wireless networks, live camera feeds, video processing, control feedback and contract-controlled coordination between UAVs. We ran an experiment on the second version of the UAV software, using the CORBA A/V Streaming Service over UDP, to informally evaluate the effectiveness of the adaptability strategy in focusing available resources when they are constrained.   We ran a total of three runs:

- A control run, with no adaptation
- A second run, where adaptation is implemented by frame dropping and
- A third run, which utilized both frame dropped and RSVP bandwidth management.

The trials were run with the sender and distributor on the same Pentium III 933 MHz processor and 512 MB of RAM, and with the receiver on a separate laptop, with a Pentium II 200 MHz processor and 144 MB of RAM, all running Linux, with a 10 Mbps link between them. We started the video running. After 60 seconds, we applied a load to the network link for 60 seconds then removed the load. After another 180 seconds, 300 seconds in all, the experiment terminated. The data collector recorded each MPEG I frame (2 per second, 600 in all) that was sent from the sender, and each that was received at the receiver, and the time elapsed from send to receive. The results of each test run are described below.

Test run 1, which as the control run without any adaptation, lost 119 of the 121 I frames sent while the system was under load, i.e., only 481 of the 600 I frames sent made it through. The average delay of the frames that made it through was 56.58 ms, with a median delay of 55 ms, a minimum delay of 38 ms and a maximum delay of 121 ms. The average delay of the frames sent through when the system was not under load was 56.33 ms, with a median of 55 ms, a minimum of 38 ms and a maximum of 67 ms. 1.65 percent of the I frames sent made it through when the system was under load (2 out of 121),

with 98.35% of the I frames being lost by the UDP transport. The average delay of the two that did make it through under load was 115.5 ms.
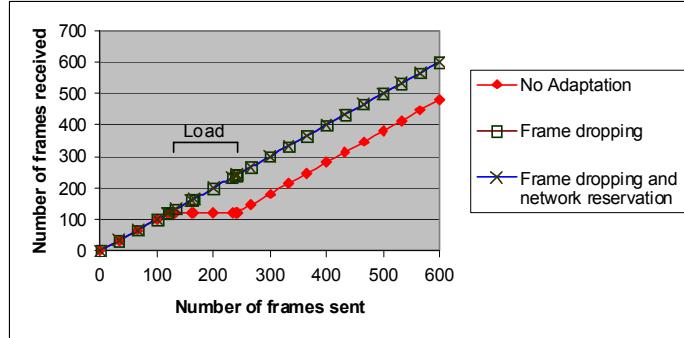
Figure 2.15: QuO adaptation ensured successful delivery of all video under load

Test run 2, with frame dropping as its only adaptation, got all 600 of its I frames through despite the load on the system. The average delay of all the frames was 70.01 ms, with a median delay of 57 ms, a minimum delay of 50 ms and a maximum delay of 143 ms. The average delay of the frames sent through when the system was not under load was 57.01 ms, with a median of 55 ms, a minimum of 50 ms and a maximum of 68 ms. 100 percent of the I frames sent made it through when the system was under load (120 out of 120), with 0% of the I frames being lost by the UDP transport. The average delay of the frames delivered while the system was under load was 122.15 ms. Test run 3, with adaptation using both frame dropping and network reservation (RSVP), also got all 600 of its I frames through despite the load on the system. The average delay of all the frames was 64.2 ms, with a median delay of 59 ms, a minimum delay of 52 ms and a maximum delay of 106 ms. The average delay of the frames sent through when the system was not under load was 58.1 ms, with a median of 56 ms, a minimum of 52 ms and a maximum of 71 ms. 100 percent of the I frames sent made it through when the system was under load (120 out of 120), with 0% of the I frames being lost by the UDP transport. The average delay of the frames delivered while the system was under load was significantly lower than the other two runs, 88.5 ms. Figure 2.15 illustrates the improvements afforded by the adaptation under load. The test runs that included QuO adaptation were able to recover from the load imposed on the system to keep the video flowing and not lose any important (i.e., I) frames. The video stream that did not have adaptive control lost nearly all the video sent during the time when load was imposed on the system.

2.8.   Open Research Issues: Strategies and Tactics for Developing Adaptive COTS Middleware with Real-time Attributes

Quorum and QuOIN have addressed many QoS aspects during the past 3 plus years. However, many key open research issues remain. Among them are the following:

Although some operating systems, networks and protocols now support real-time scheduling, they do not provide integrated end-to-end solutions. In particular, QoS research on networks and operating systems has not addressed key requirements and end-to-end usage characteristics of mission-critical real-time systems developed using COTS middleware, such as CORBA. The solutions developed

have generally focused on either specific signaling and enforcement mechanisms or broadly based resource allocation techniques. These approaches have not focused on providing both a vertically (i.e., network interface ⇔ application layer) and horizontally (i.e., end-to-end) integrated solution that provides a higher level service model, or global policy framework, to developers and end-users. For instance, research on QoS for ATM networks has focused largely on policies and mechanisms for allocating network bandwidth on a per-connection basis. Recent research on Internet2 topics has developed resource allocation signaling protocols, such as RSVP, and global resource sharing techniques, such as Differentiated Services. Likewise, research on real-time operating systems has focused largely on avoiding priority inversions and non-determinism in synchronization and scheduling mechanisms for multi-threaded applications. While these research activities are important building blocks, they tend to yield point solutions that emphasize relatively fixed or static policies and mechanisms.

Introducing application-level awareness of changes to expected and delivered QoS is a new direction for inserting adaptive behavior into distributed applications. Adaptation can occur at any and all of the various layers of the system, including customized approaches in the application itself and standard service (re) configurations within the supporting middleware infrastructure. An example of an application-level adaptation might be moving from full motion video to audio and still imagery to text-only interactions. An example of a service-level adaptation might be acquiring additional bandwidth by preempting a lower priority user or automatically instantiating additional resource replicas when another one becomes unreachable. The keys to success in these adaptations lies in developing paths through the system layers that can effectively coordinate the otherwise independent activities so they provide maximum utility to developers and users without conflicting behavior that might result from a series of independent or transparent actions.

Determining how to map the results from existing QoS research on global policies and local enforcement techniques onto adaptive real-time COTS middleware is an important open research issue. Thus, to meet the research challenges and resolve the key design issues it is necessary to ensure that the benefits of each independent research area are preserved in the resulting architectural framework, while relaxing some of the assumptions that are specific to the context in which the results were obtained. In addition, some of the resource management strategies that are effective in small, self-contained environments will likely require extensions or alternate approaches for more open-ended environments of significantly larger scale. There are continuing investigations in this area.

2.9. Concluding Remarks

Adaptive COTS middleware is a promising solution for some key challenges facing researchers and developers of distributed and embedded real-time mission-critical systems. However, meeting the QoS requirements of the next-generation of these systems requires more than higher-level design and programming techniques, such as encapsulation and separation of concerns, associated with conventional COTS middleware. In addition, it requires an integrated architecture, based on adaptive middleware that can deliver end-to-end QoS support at multiple levels in distributed and embedded real-time systems.

Supporting this adaptive middleware architecture effectively requires powerful dynamic and adaptive resource management techniques that extend existing static resource management techniques. By preserving the key capabilities of the static approaches, and generalizing those capabilities to include

dynamic and hybrid static/dynamic capabilities, adaptive real-time mission-critical systems can be built to address the needs of a broad class of applications. The key is to support a multi-dimensional end-to-end QoS framework that allows middleware and application developers to more easily control and coordinate the collection of lower-level mechanisms that come into play, using techniques that are simple to use, understand and validate. The Quorum program and QuOIN integration activity has provided a running start on this agenda.

2.10. References

[1] D. Box, "Essential COM", Addison-Wesley, Reading, MA, 1997

[2] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad and Michael Stal, "Pattern-Oriented Software Architecture- A System of Patterns", Wiley and Sons, 1996

[3] M. Cukier, J. Ren, C. Sabnis, D. Henke, J. Pistole, W. Sanders, D. Bakken, M. Berman, D. Karr and R. E. Schantz, "AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects ", *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, pages 245-253, October 1998

[4] DARPA, The Quorum Program, http://www.darpa.mil/ito/research/quorum/index.html, 1999

[5] B.S. Doerr, and D.C. Sharp, "Freeing Product Line Architectures from Execution Dependencies", *Proceedings of the 11th Annual Software Technology Conference*, 1999

[6] B.S. Doerr, T. Venturella, R. Jha, C.D. Gill, and D.C. Schmidt, "Adaptive Scheduling for Real-time, Embedded Information Systems", *Proceedings of the 18th IEEE/AIAA Digital Avionics Systems Conference (DASC),* 1999

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, MA, 1995

[8] Christopher Gill et al., "Applying Adaptive Real-time Middleware to Address Grand Challenges of COTS-based Mission-Critical Real-Time Systems", *Proceedings of the 1st IEEE International Workshop on Real-Time Mission-Critical Systems: Grand Challenge Problems*, November 1999

[9] C.D. Gill, D.L. Levine, and D.C. Schmidt, "The Design and Performance of a Real-Time CORBA Scheduling Service", *The International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware*, to appear 2000

[10] Tim Harrison, David Levine and Douglas Schmidt, "The Design and Performance of a Real-time {CORBA} Event Service", *Proceedings of OOPSLA '97*, Atlanta, GA, October 6-7, 1997

[11] T.H. Harrison, C. O'Ryan, D.L. Levine, and D.C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service", *Journal on Selected Areas in Communications, special issue on Service Enabling Platforms for Networked Multimedia Systems*, 17(9), 1999

[12] Gregor Kiczales, John Irwin, John Lamping, Jean-Marc Loingtier, Cristina Videria Lopes, Chris Maeda, and Anurag Mendhekar, "Aspect-Oriented Programming", *ACM Computing Surveys*, 28(4), December 1996

[13] F. Kuhl, R. Weatherly, and J. Dahmann, "Creating Computer Simulation Systems", Prentice Hall, Upper Saddle River, New Jersey, 1999

[14] D.L. Levine, C.D. Gill, and D.C. Schmidt, "Dynamic Scheduling Strategies for Avionics Mission Computing", *Proceedings of the 17th IEEE/AIAA Digital Avionics Systems Conference (DASC),* 1998

[15] J.P. Loyall, R.E. Schantz, J.A. Zinky, and D.E. Bakken, "Specifying and Measuring Quality of Service in Distributed Object Systems", *Proceedings of The 1st IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 98)*, 1998

[16] Joseph P. Loyall, David E. Bakken, Richard E. Schantz, John A. Zinky, David Karr, Rodrigo Vanegas and Ken R. Anderson, "QoS Aspect Languages and Their Runtime Integration", *Proceedings of the Fourth Workshop on Languages, Compilers and Runtime Systems for Scalable Components*, May 1998

[17] Joseph P. Loyall, Partha P. Pal, Richard E. Schantz and Franklin Webber, "Building Adaptive and Agile Applications Using Intrusion Detection and Response", *Proceedings of the ISOC Network and Distributed Systems Security Conference*, to appear Februrary, 2000

[18] Object Management Group, "Fault Tolerance CORBA Using Entity Redundancy RFP", OMG Document orbos/98-04-01 edition, 1998

[19] Object Management Group, "Realtime CORBA Joint Revised Submission", OMG Document orbos/99-02-12 edition, 1999

[20] Object Management Group, " The Common Object Request Broker: Architecture and Specification CORBA/IIOP 2.3, OMG Technical Document 98-12-01", December 1999, Framingham, MA

[21] C. O'Ryan, D.C. Schmidt, and D. Levine, "Applying a Scalable CORBA Events Service to Large-scale Distributed Interactive Simulations", *Proceedings of the 5th Workshop on Object-oriented Real-time Dependable Systems*, Montery, CA, 1999

[22] Partha Pal, Joseph Loyall, Richard Schantz, John Zinky, Rich Shapiro and James Megquier, "Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration", *Proceedings of The 3rd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 00)*, to appear March 2000

[23] Prashant Chandra, "Darwin: Resource Management for Value-Added Customizable Network Service", *Sixth IEEE International Conference on Network Protocol (ICNP'98)*, Austin, TX, 1998

[24] Pyarali, C. O'Ryan, D.C. Schmidt, N. Wang, V. Kachroo, and A. Gokhale, "Applying Optimization Patterns to the Design of Real-time ORBs", *Proceedings of the 5th Conference on Object-Oriented Technologies and Systems*, San Diego, CA, 1999

[25] C. Sabnis, M. Cukier, J. Ren, P. Rubel, W.H. Sanders, D.E. Bakken and D.A. Karr, "Proteus: A Flexible Infrastructure to Implement Adaptive Fault Tolerance in AquA", *Proceedings of the 7th IFIP Working Conference on Dependable Computing for Critical Applications (DCCA-7),* pp137-156, San Jose, CA, January 1999

[26] Richard E. Schantz, John A. Zinky, David A. Karr, David E. Bakken, James Megquier and Joseph P. Loyall, "An Object-level Gateway Supporting Integrated-Property Quality of Service", *Proceedings of The 2nd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 99)*, May 1999

[27] D.C. Schmidt and C. Cleeland, "Applying Patterns to Develop Extensible ORB Middleware", *IEEE Communications Magazine*, 37(4), 1999

[28] D.C. Schmidt, T.H. Harrison, and E. Al-Shaer, "Object-Oriented Components for High-speed Network Programming", *Proceedings of the 1st Conference on Object-Oriented Technologies and Systems*, Monterey, CA, 1995

[29] D.C. Schmidt, D.L. Levine, and S. Mungee, "The Design and Performance of Real-Time Object Request Brokers", *Computer Communications* 21(4), pp. 294—324, 1998

[30] D.C. Schmidt, S. Mungee, S. Flores-Gaitan, and A. Gokhale, "Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers", *Journal of Real-time Systems*, to appear 2000

[31] R. Steinmetz, "Synchronization Properties in Multimedia Systems", *Journal on Selected Areas in Communications* 8(3), 1990

[32] Daniel F. Sterne, Gregg W. Tally, C. Durward McDonell, David L. Sherman, David L. Sames, Pierre X. Pasturel and E. John Sebes, " Scalable Access Control for Distributed Object Systems ", *Proceedings of the 8th Usenix Security Symposium*, August 1999

[33] Sun Microsystems, "Jini Connection Technology", http://www.sun.com/jini/index.html 1999

[34] Anne Thomas, "Enterprise JavaBeans Technology", http://java.sun.com/products/ejb/white_paper.html, December 1998

[35] Rodrigo Vanegas, John A. Zinky, Joseph P. Loyall, David Karr, Richard E. Schantz and David E. Bakken, "QuO's Runtime Support for Quality of Service in Distributed Objects", *Proceedings of Middleware 98, the IFIP International Conference on Distributed Systems Platform and Open Distributed Processing*, September 1998

[36] Partha Pal, Joseph Loyall, Richard Schantz and Franklin Webber, "Open Implementation Toolkit for Building Survivable Applications", *DARPA Information Survivability Conference and Exposition (DISCEX) 2000*, January 25 - 27, 2000, Hilton Head Island, SC

[37] A. Wollrath, R. Riggs, and J. Waldo, "A Distributed Object Model for the Java System", *USENIX Computing Systems*, 9(4), 1996

[38] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network*, September 1993

[39] Wei Zhao and Satish K. Tripathi, "A Resource Reservation Scheme for Synchronized Distributed Multimedia Sessions", *1st Annual Advanced Telecommunications/Information Distribution Research Program Conference*, January 21-22, 1997

[40] John A. Zinky, David E. Bakken, and Richard E. Schantz, "Architectural Support for Quality of Service for CORBA Objects", *Theory and Practice of Object Systems* 3(1), 1997

[41] Karr D.A., Rodrigues C., Loyall J.P., Schantz R.E., Krishnamurthy Y., Pyarali I., Schmidt D.C. "Application of the QuO Quality-of-Service Framework to a Distributed Video Application," *Proceedings of the International Symposium on Distributed Objects and Applications*, September 18-20, 2001, Rome, Italy

[42] Corman, David, October 2001, *WSOA–Weapon Systems Open Architecture Demonstration–Using Emerging Open System Architecture Standards to Enable Innovative Techniques for Time Critical Target (TCT) Prosecutio*n, 20th Digital Avionics Systems Conference (DASC), Daytona Beach, Florida, IEEE/AIAA.

[43] Karr DA, Rodrigues C, Loyall JP, Schantz RE, Krishnamurthy Y, Pyarali I, Schmidt DC. Application of the QuO Quality-of-Service Framework to a Distributed Video Application. Proceedings of the International Symposium on Distributed Objects and Applications, September 18-20, 2001, Rome, Italy.

[44] Loyall JL, Gossett JM, Gill CD, Schantz RE, Zinky JA, Pal P, Shapiro R, Rodrigues C, Atighetchi M, Karr D. Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications. Proceedings of 21st IEEE Intern'l Conference on Distributed Computing Systems (ICDCS-21), April, 2001, Phoenix, AZ.

[45] Huang, Jim Jiandong; Jha, R.; Muhammad, Mustafa; Lauzac, S.; Kannikeswaran, B.; Schwan, K.; Zhao, W.; Bettati, R. RT-ARM: a real-time adaptive resource management system for distributed mission-critical applications. IEEE Workshop on Middleware for Distributed Real-time Systems and Services, 2 December 1997, San Francisco, CA.

[46] Christopher D. Gill, David L. Levine, and Douglas C. Schmidt The Design and Performance of a Real-Time CORBA Scheduling Service, Real-Time Systems: the International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware, guest editor Wei Zhao, March 2001, Vol. 20 No. 2

[47] OMG. *Control and Management of Audio/Video Streams, OMG RFP Submission (Revised), OMG Technical Document 98-10-0*5. Object Management Group, Framingham. MA, Oct 1998.

3. Dependability as a QoS Aspect: Detailed Design and Implementation

AQuA Dependability Management

University of Illinois investigators:

William H. Sanders (PI), Ramesh Chandra, Tod Courtney, Michel Cukier, Harpreet S. Duggal, David Henke, Kaustubh Joshi, Sudha Krishnamurthy, Ryan M. Lefever, Jessica Pistole, Yansong (Jennifer) Ren, Paul Rubel, Chetan Sabnis, and Mouna Seri

3.1. Background and Summary

This section describes the technical accomplishments and impacts of the AQuA Project, which was supported by a subcontract to the University of Illinois. The AQuA project builds on previous work which was funded in part by two previous DARPA contracts within the Quorum program, as well as by UIUC institutional funds. The project has succeeded in creating an adaptable simple-to-use middleware for creating dependable CORBA-based distributed applications that must satisfy particular dependability and soft real-time constraints.

Using the framework, a user is able to make standard CORBA distributed applications dependable by adding high-level QoS-request calls to specify the desired dependability and real-time characteristics of remote method invocations, and by using the AQuA gateway and AQuA dependability manager to manage the application at run-time. These two key components manage the in-band remote method invocations that an application makes and its desired quality of service, respectively. In particular, the AQuA gateway, a major innovation of the project, acts as a smart proxy for remote objects, managing request invocations in a way that provides the desired quality of service to an application. Its actions include sending the request to a set of remote method replicas to provide dependability or probabilistic real-time guarantees and managing crash, value, and timing faults in a nearly transparent way with respect to the application. The AQuA dependability manager reacts to changes in the desired quality of service of a set of applications and to crash, value, and timing faults that occur, reconfiguring the system to maintain the desired quality of service level.

The developed middleware has been demonstrated on numerous occasions, and has been transferred and used at the US Government Naval Surface Warfare Center, Combat Systems Technologies Branch (Dahlgren Division), and at Boeing Corporation (in St. Louis). This use has provided us with valuable feedback, and enabled us to make significant improvements to the usability of the software. Given this use and feedback, we now plan to make an open-source release of the AQuA middleware, and make it available to other academic and industrial users. In its current form, the AQuA middleware is mature, is ready to be used by others, and greatly simplifies the construction of dependable distributed applications.

3.2. Motivation

Providing fault tolerance to distributed applications is a challenging and important goal. In many applications, the cost of a custom hardware solution is prohibitive. Even if custom hardware is used, the flexibility that software can provide makes it a natural choice for implementing a significant portion of the fault tolerance of dependable distributed systems. Furthermore, when the dependability and soft real-time requirements change during the execution of an application, the fault tolerance approach must be adaptive in the sense that the mechanisms used to provide fault tolerance may change at runtime. Together, these requirements argue for a software solution that can reconfigure a system based both on the levels of dependability and soft real-time constraints desired by a distributed system during its execution and on the faults that occur.

The AQuA architecture provides a flexible approach for building dependable, distributed, object-oriented systems that support adaptation in response to both faults and changes in an application's dependability and soft real-time requirements. It provides a simple high-level way for applications to specify the levels of dependability and soft real-time constraints they desire and the type of faults that should be tolerated.

The AQuA dependability manager (which we reference as Proteus) provides fault tolerance in AQuA by dynamically managing replicated distributed objects to make them dependable. It does this by configuring the system in response to faults and changes in desired dependability and soft real-time requirements. The choice of how to provide fault tolerance involves choosing the types of faults to tolerate, the styles of replication to use, the degrees of replication to use and the location of the replicas, among other factors. The replication protocols in Proteus assume the existence of an underlying group communication system that provides reliable multicast, total ordering and virtual synchrony [Bir96]. For our implementation, we have used the Maestro/Ensemble [Bir96, Hay98, Vay98] group communication system. Communication between all architecture components is done using gateways, which translate CORBA object invocations into messages that are transmitted via Ensemble, and contain mechanisms to implement a chosen fault tolerance scheme.

Several other projects have similar aims. These projects can be classified into three categories. The first approach is to create a fault-tolerant ORB. Both Electra [Maf95, Maf97] and Maestro fall into this category. The second category involves providing fault tolerance through a CORBA service, above the CORBA Object Request Broker (ORB). The OpenDREAMS [Fel96] project and Arjuna [Mor99] take this approach. A third method is to intercept messages from the ORB; this is the approach taken by Eternal [Mos98, Nar97, Nar99b, Nar00, Nar01].

3.3. AQuA Description

The AQuA architecture [Ren01a, Ren01b, Rub00] is a framework for building dependable, distributed, object-oriented systems that support adaptation to both faults and changes in an application's dependability requirements. It was developed concurrently with, but independently from, the CORBA fault tolerance standard. It provides the types of fault tolerance specified by the standard. Moreover, AQuA currently is able to provide services that are not specified in the fault tolerance standard, such as support for any standard CORBA applications, without the limitation of requiring the replicated server objects to use the same ORB, and including support for protection against value faults in the body of the IIOP message.
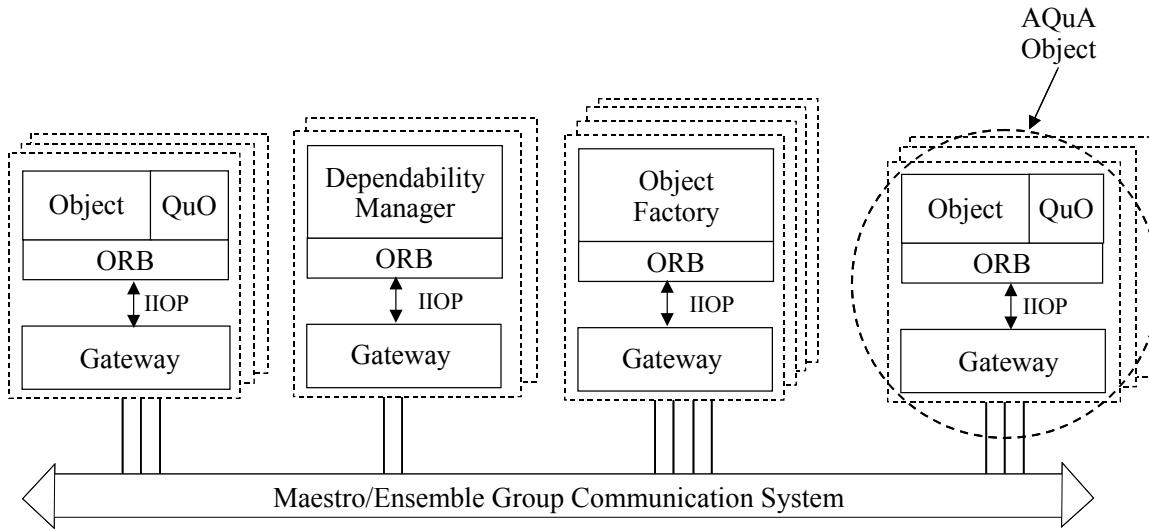
Figure 3.1: Overview of the AQuA Architecture

Figure 3.1 shows the different components of the AQuA architecture in one particular configuration. These components can be assigned to hosts in many different ways, depending on an application's desired level of dependability.

In AQuA, fault tolerance is achieved through the replication of objects. All replicas of an object form a group. Messages communicated among different objects are sent through groups. To provide fault tolerance at the most basic level, the AQuA system uses the Ensemble group communication system: to ensure reliable communication between groups of processes, to ensure that totally ordered messages are delivered to the members in a group, to maintain group membership based on the virtual synchrony model and to detect and exclude from the group members that fail by crashing. Ensemble assumes that process failures are fail-silent (or crash failures), and detects process failures through the use of "I am alive" messages. The AQuA architecture uses this detection mechanism to detect crash failures, and provides input to Proteus to aid in recovery.

An application can specify its dependability requirements via a Quality Object (QuO) [Loy98a, Loy98b, Zin97]. It allows distributed applications to process and invoke dependability requests, and to receive information regarding the level of dependability that is being provided by the current system. QuO allows distributed object-oriented applications to specify dynamic QoS requirements. In the AQuA approach, QuO is used to transmit applications' dependability requirements to Proteus, which attempts to configure the system to achieve the desired level of dependability. QuO also provides an adaptation mechanism that is used when Proteus is unable to provide the specified level of dependability or meet the specified soft real-time constraints.

In AQuA, Proteus provides adaptive fault tolerance. It consists of a replicated dependability manager, a set of object factories and gateway handlers. The dependability manager determines a system configuration based on reports of faults and desires of application objects. An object factory that

resides on each host is used to create and kill objects, as well as to provide load and other information about the host to the dependability manager.

Communication between all architecture components (i.e., applications, the QuO runtime, object factories and dependability managers) is done using gateways, which translate CORBA object invocations into messages that are transmitted via Maestro/Ensemble. Furthermore, the handlers in a gateway implement multiple replication schemes and communication mechanisms. The handlers are also used to detect application value faults, and to report value faults and group membership changes to the dependability managers. Before discussing the AQuA group structure and gateway architecture, we review Proteus.

3.4. Proteus Overview

Most group communication systems, including Ensemble, are based on the assumption that processes fail by crashing, but no mechanism is implemented to ensure that processes fail only by crashing. Furthermore, recovery by automatic start of new processes on the same or different hosts is not implemented in the protocol stack. Instead, it is left to the application. A fault tolerance framework is thus necessary to tolerate other fault types and provide recovery mechanisms that are more sophisticated than process exclusion. The framework could be implemented at the process level through an implementation of further fault tolerance in Ensemble. However, in order to be independent of any particular group communication system and to fully use the features offered by CORBA applications, we have provided additional fault tolerance above the group communication infrastructure. The framework we have developed is able to tolerate crash failures of processes and hosts, as well as value faults and timing failures of CORBA objects. In addition to the fault tolerance mechanisms themselves, two types of replication can be used: active and passive. Active replication includes pass-first, leader-only and majority voting schemes. Passive replication includes stable storage and state cast schemes. Finally, a replication scheme combining both replication types is used for managing soft real-time constraints.

3.5. Groups in the AQuA Architecture

In AQuA, we use a general object model rather than the more restrictive client/server model. The model of computation is thus based on interactions between objects that can be replicated. Objects can initiate requests (acting as clients) and respond to requests (acting as servers). In the AQuA architecture, the basic unit of replication is a two- or three-process pair, consisting of either an application and gateway or application, gateway and QuO runtime. A QuO runtime is included if an object contained in the application process makes a remote invocation of another object and wishes to specify a quality of service for that object. A basic replication unit may contain one or more distributed objects, but to simplify the following discussion, we refer to it as an "AQuA object." Furthermore, when we say that an "object joins a group" we mean that the gateway process of the object joins the group. Mechanisms are provided to ensure that if one of the processes in the object crashes, the others are killed, thus allowing us to consider the object as a single entity that we want to make dependable.

Using this terminology, we can now describe the group structure and mechanism used in the AQuA architecture, including replication groups and connection groups. By defining multiple replication and connection groups, we can avoid the communication overhead that would occur if a single large group

were used. A replication group is composed of one or more replicas of an AQuA object. These objects may be transient or persistent members of the group. Persistent members join the group when they are created, and remain in the group. Transient members join a replication group only when they need to multicast a message to the replicas in the group. After sending a message, these objects leave the group. A replication group has one persistent object that is designated as its leader and may perform special functions. Each persistent object in the group has the capacity to become the object group leader, and a protocol is provided to ensure that a new leader is elected when the current leader fails.

A connection group is a group consisting of the persistent members of two replication groups that wish to communicate. It provides reliable message communication from one CORBA object to another CORBA object.

As specified above, each replicated object is located inside a replication group. AQuA provides two methods of reliable communication between objects that are in two different replication groups. One way is to use a connection group, which is done if the sending object is a persistent member of its replication group, and hence is in a connection group shared by the destination replicated object. In that case, a replicated object that is inside a replication group multicasts messages within a connection group to forward them to the other replicated object. Using that approach, two different objects are able to communicate using both one-way and synchronous remote method invocations. This approach requires that there be a pre-established connection group before objects send messages to each other. In the second method of reliable communication, the sending object becomes a transient member of a replication group with which it wishes to communicate. The invocations made by transient members can only be one-way. In addition, only the leader of a sender replication group is allowed to become a transient member of another replication group, and the leader is responsible for making invocations on behalf of the sender replication group. The method is only suitable for situations in which duplicate messages are allowable (at-least-once semantics). Communication through a transient group member is useful in situations in which communication is fairly infrequent. In such cases, the overhead in joining and leaving a replication group is small relative to that of maintaining a connection group between two replication groups.

Consider Figure 3.2 for an illustration of the possible use of replication groups and connection groups. Solid lines define the replication and connection groups. The dashed oval represents the occurrence of a transient member joining a replication group. We see in Figure 3.2 that even though a connection group is composed of two replication groups, a member of a replication group can be included in several connection groups. For example, the replicas in replication group 3 communicate with the replicas in replication group 1 through connection group 1, and they communicate with the replicas in replication group 2 through connection group 2. The leader of replication group 2 becomes a transient group member of replication group 1 in order to send messages to the replicas in replication group 2.
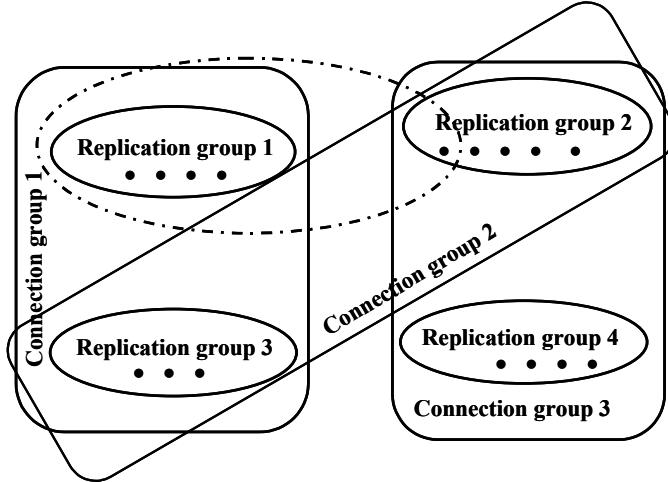
Figure 3.2: Example Group Structure in AQuA

3.6. AQuA Gateway

The AQuA gateway is a process that is associated with each CORBA application. The AQuA gateway provides fault tolerance by implementing different communication schemes and replication protocols. These fault-tolerance mechanisms provide reliable remote method invocations, no matter when and where server object replicas fail before the client receives the replies. To achieve this dependability, each CORBA client's invocation is forwarded by its gateway to a set of replicated CORBA server objects, and only one copy of the reply message is allowed to return to the client object. The AQuA gateway is responsible for finding a set of replicated objects that can implement the request, passing them the parameters, invoking their methods and returning the results. The client does not know where the server objects are located, or how many replicated server objects process the invocations.

Figure 3.3 shows the physical structure of a gateway. It contains a gateway ORB, a naming service, a handler factory, a set of handlers and a DII processor. The gateway ORB is a standard ORB (the TAO ORB [TAO] is used in our implementation). It works as a normal ORB to communicate with standard CORBA applications using IIOP messages. In that way, the AQuA gateway is able to communicate with different commercial ORBs to provide ORB transparency.
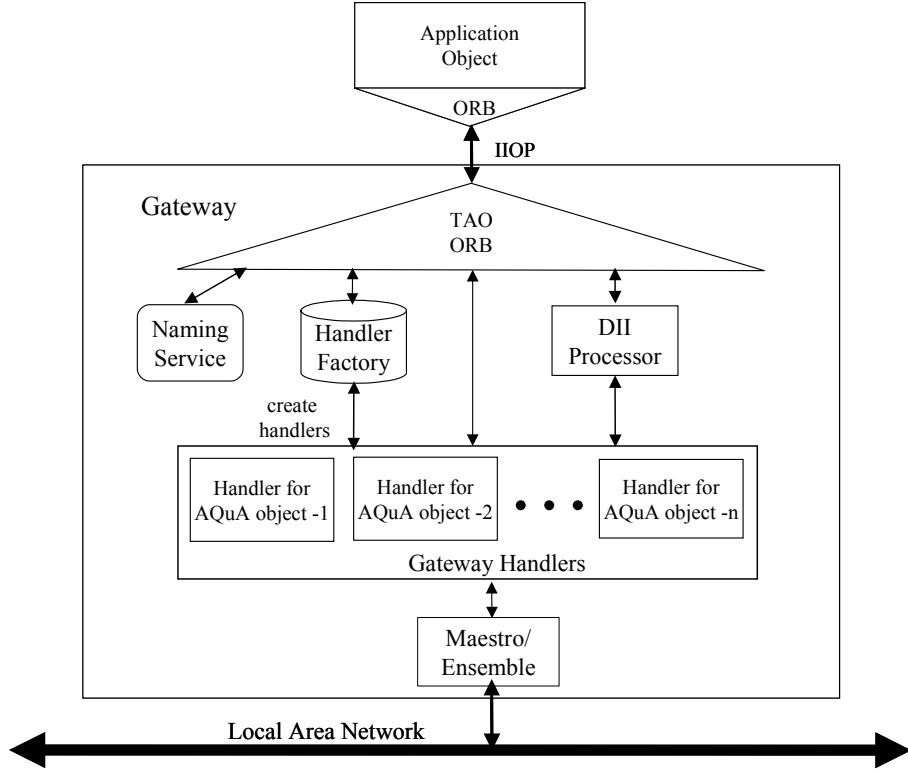
Figure 3.3: AQuA Gateway

The naming service maps object names to object references. It is a local naming service in a gateway, and provides a way for the CORBA application to communicate with the gateway handlers. In particular, in the client gateway, the naming service associates the remote server objects' names with the handlers' object references to direct all of the client's invocations to the gateway handlers instead of allowing them to go directly to remote server objects. In the server gateway, the naming server helps the gateway handlers locate the server objects.

The handler factory is responsible for creating handlers. It includes a handler repository that provides a set of different types of handlers. The handlers can be classified into two categories: static and dynamic. Static handlers have persistent group memberships. These handlers remain in the appropriate Ensemble process groups once they have been created and joined the groups. They are used to implement replication schemes, and are therefore also called "replication handlers." The replication handlers can be further classified into active replication, passive replication and soft-realtime handlers. The active replication handlers are used to implement the active replication schemes. They include: the pass-first handler, the leader-only handler and the majority-voting handler. The passive replication handlers are used to implement the passive replication schemes. They include the state cast handler and the stable storage handler.

Dynamic handlers, the second category of handlers, are transient replication group members. Each handler is responsible for sending and receiving messages for a particular replicated object. When a gateway handler receives an invocation, it will first remove the IIOP header, and then

construct a gateway message that will carry the CORBA invocation to the gateways of the remote replicated servers. Each gateway message includes two parts: a message header and a payload of a CORBA IIOP message. The message header contains information used by replication schemes to process and deliver messages correctly. Each header has several fields: sender, receiver, is_oneway, sequence number, ID and opcode. Each field has a special purpose related to the communication scheme. The sender and receiver fields specify the source and destination of an invocation. The is_oneway field indicates whether the invocation is an asynchronous (one-way) or synchronous CORBA message. The sequence number is an integer that is assigned to each invocation and is uniquely associated with a sender and receiver pair. The ID indicates which replica generated the message. The opcode is used for implementing replication steps.

After constructing a gateway message, the handler will encapsulate it into a Maestro/Ensemble group communication message so that it can be sent over the network to the replicated server gateways. If an invocation is a synchronous CORBA message, the handler is also responsible for receiving the reply from the group communication system. In the replicated server gateways, once a handler receives a group communication message, it will first unencapsulate the received Maestro message to get the gateway message. It will then remove the gateway header and get the name of the operation and the arguments for the original IIOP invocation from the gateway message payload. Next, it will construct a new dynamic invocation based on the information from the gateway message, and forward the invocation to the DII processor.

The DII processor is used to deliver invocations received from the handlers to the application object. In order to ensure strong data consistency among replicas, the DII processor contains a synchronous queue that ensures that the incoming invocations are delivered to the application in the order in which they were received from the group communication system. If the invocation is a synchronous CORBA message, the DII processor will wait for a reply, and then return the reply to the server gateway handler that is responsible for forwarding the reply to the client object. Since all of the replicated server objects generate replies, the server gateway handlers will allow only one copy of the replies to be sent back to the handler servant in each client's gateway, to ensure that there are no duplicate replies.

3.7. QoS Requests / Programming Interface

An application or QuO programmer interacts with the Proteus dependability manager (1) to request a particular level of dependability and soft real-time constraints, (2) to be notified when that level is no longer met, (3) to obtain information concerning hosts managed by Proteus and give advice about which hosts the dependability manager should place replicas on and (4) to obtain detailed information regarding decisions that the dependability manager makes and the faults that it detects. The interface to the dependability manager can be divided into two sets of methods: those used to communicate with the components in the AQuA system core (composed of the dependability manager, the gateway handlers and the object factories), and those used by one or more AQuA objects to request and observe QoS, to observe the state of the dependability manager and to observe and control hosts. Proteus supports the development of three types of objects that can make QoS requests from the dependability manager and also observe its actions. One of these object types, called the "QoS observer/requester," can be used to make QoS requests to the dependability managers and can receive callbacks regarding the ability of the dependability manager to satisfy the requester's requests. (An example of an application that may contain a QoS observer/requester is QuO itself.) Furthermore, since the

dependability manager supports a standard, well-defined interface, an application object can also make QoS requests directly to the dependability manager. The second type of object the dependability manager supports is advisor observers. Advisor observers can "subscribe" to a variety of information used by the dependability manager to make decisions, including information about faults detected and fine-grained information regarding actions taken by the manager. Proteus also supports the development of a third type of objects, called "host observer/controllers," that receive information regarding the status of hosts that may be used to execute object replicas, and that can be used to specify particular hosts for the execution of replicas. In particular, host observers/controllers can be used by an application or QuO to specify hosts that should not be used to execute replicas, if the application or QuO has information that leads it to believe that the host should not be used.

3.8. *Replication Protocols Supported*

In AQuA, five types of replication techniques have been developed: active replication with pass-first, active replication with majority voting, active replication with leader-only, passive replication and one combination of replication types for providing soft real-time constraints. The different replication techniques enable the tolerance of different types of faults, provide different recovery strategies, use different communication schemes and support different types of object replicas (deterministic or nondeterministic). On the other hand, all of the replication techniques must provide the ability to (1) ensure reliable transmission of each CORBA message from one replicated object to another so that messages will not be lost even if replicas crash, (2) guarantee strong data consistency among all object replicas, (3) guarantee that no duplicate messages are delivered to the replicated objects and (4) react correctly when the number of replicas in a replication group changes. In addition, all of the replication techniques require the use of the group communication system to provide reliable multicast, totally ordered message delivery, group membership services and virtual synchrony. The types of replication techniques can be divided into three categories: active replication, passive replication and a combination of replication types.

3.8.1. Active Replication

In active replication, all members of the object group independently execute the methods invoked on the object, so that if a fault prevents one member from operating correctly, the other members will produce the required messages. This technique requires that the CORBA ORB and the object replicas be deterministic to ensure that incoming messages are dispatched from the ORB to the replicas in the same order, and that the replicas process the messages in the same order. There are three types of active replication schemes:

1. Active Replication with Pass-First Scheme: In this scheme, each replica in the replication group executes each invocation independently and sends each request/reply to the leader of the group. The leader is responsible for forwarding the first received request/reply to the destination object group. This scheme can be used to tolerate process crash failures.
2. Active Replication with Leader-Only Scheme: In this scheme, the leader processes input messages and sends its output messages to the destination object group. The other members will process input messages and generate output messages that are suppressed unless the member takes over for the leader (if the leader fails). Through use of this scheme, process crash failures can be tolerated.
3. Active Replication with Majority Voting Scheme: In this scheme, each replica in the replication group executes each invocation independently and multicasts its request/reply in the replication group. The requests/replies from the members of the source object group is voted on; if and only if

a majority of the requests/replies are identical, the majority values are delivered to the members of the destination object group. The leader is responsible for forwarding the voting results to the destination group. Each replica executes majority-voting algorithms independently in order to take over from the leader if it fails.

In all three active replication schemes, in the event that one replica fails, the application can continue with results from another replica without waiting for fault detection and recovery.

3.8.2. Passive Replication

In passive replication, during fault-free operation, only one member of the object group, the leader, executes the method invoked on the group. The gateways of the other replicas store the sequence of method invocations in a buffer but do not deliver the messages to their applications, and thus do not process the request. Periodically, the state of the leader is transferred to the other members (the backup replicas) or to stable storage. In the presence of a leader crash, a backup member becomes the new leader of the group and updates its state. The state of the new leader is made identical to the state of the old leader through application of the request messages recorded in the new leader's gateway buffer. Passive replication can be used to tolerate process crash failures.

There are two types of passive replication schemes: the passive replication with state cast scheme and the passive replication with stable storage scheme. In the first scheme, the leader multicasts its state to the backup replicas. This scheme provides a way for the backup replicas to access the state immediately during fault recovery. However, it requires that more messages be transmitted over the network. In the second scheme, the leader stores its state in stable storage. When the original leader fails, the new leader will take over from the original leader by getting the state from stable storage. Compared to the state cast scheme, this scheme reduces the number of messages transmitted over the network. However, the recovery time could be longer than for the state cast scheme, because the new leader needs time to get the state from stable storage. The extra recovery time includes both the time required for disk access and the time spent on the network.

In both of the passive replication schemes, two strategies: every-message and periodic, can be used to capture the state of the leader. Every-message state transfer happens whenever the leader sends a message to the outside world. For that reason, output messages will not need to be resent if the leader already sent them out before it failed. Thus, with every-message state transfer, replicas are not required to be deterministic. Periodic state transfer occurs when the leader sends out a certain number of messages. In that scenario, if the leader fails, the backup replica that takes over for it might send out duplicate messages that must be suppressed. Thus, with periodic state transfer, replicas are required to be deterministic, so that the new leader produces messages that are the same as those produced by the original leader before its failure.

3.8.3. Probabilistic temporal guaranties using replication

We have developed not only handlers to provide strong data consistency through active and passive replication, but also a handler for providing probabilistic temporal guaranties using replication. The goal of this handler is to prevent timing failures by dynamically selecting the replicas that can satisfy a client's timing requirement, even when the quality of service is degraded due to replica failures and excess load on the server. The approach we use estimates a replica's response time distribution based on performance measurements regularly broadcast by the replica. An online model uses these measurements to predict the probability with which a replica can prevent a timing failure for a client.

A selection algorithm then uses this prediction to choose a subset of replicas that can together meet the client's timing constraints with at least the probability requested by the client.

3.9.   Conclusions and Future Work

This section of the final project report presented an overview of the AQuA middleware, which provides a flexible and extensible approach to building dependable, object-oriented distributed systems. Systems built using AQuA support adaptation to changes in system resources due to both faults in the environment and changes in an application's dependability and soft real-time requirements. The software is mature, easy-to-use and documented, making it possible for others to directly benefit from the research results from this part of the project.

AQuA provides a flexible infrastructure for providing adaptive fault tolerance to CORBA applications. Its design permits an application to change the level of dependability and soft real-time constraints that it requires, including the type of faults that should be tolerated dynamically during its execution. In order to make this possible, we have designed AQuA in a modular way, developing a scalable group structure and a set of communication algorithms that preserve needed communication properties during intergroup communication. Gateways were designed that make use of this group structure and support multiple replication and communication schemes through the use of different handlers. The implementation includes support for multiple handlers to tolerate crash, value and timing faults and a dependability manager policy that permits changing the degree of replication and placement of replicas during execution based on the dependability desires of an application. In addition, a graphical user interface for the dependability manager and object factories was developed to allow the functioning of AQuA to be monitored as it responds to dependability requests from applications and faults that occur. A user can monitor changes in membership that occur in replication and connection groups and in assignment of objects to hosts. Extensive performance measurements have been taken of the system and are documented in Yansong (Jennifer) Ren's Ph.D. thesis. The results show that AQuA has the ability to detect failures quickly, to recover from them, and to have short replica blocking times.

In addition to providing results that can be used immediately, the AQuA project provided inspiration for future work in property management in Quality-of-Service middleware. In particular, one major research direction that we are currently pursuing is the development of a middleware-based intrusion tolerance approach that helps applications survive certain kinds of attacks. Our approach is similar to that taken in AQuA, in that it makes use of a gateway and management infrastructure, but differs greatly in the details, since toleration of malicious attacks requires much more sophisticated algorithms, at all levels of the middleware (group communication, gateway, and manager), than those needed to tolerate crash, value, and timing faults. To achieve this functionality, we plan to use process- and object-level intrusion-tolerant group communication, integrate a set of COTS security tools that together with information from the group communication system itself, detect corrupt processes, and provide a decentralized replica management facility that decides what to do (in a possibly unpredictable way) when intrusions occur. In doing so, we will assume that as a result of an attack, replicas and management entities can fail in arbitrary ways.

A second major research direction we intend to pursue is the use of *on-line* models to facilitate adaptation, at multiple levels of granularity, of systems that employ quality-of-service middleware. Our work on building soft-real-time handlers in the AQuA gateway has shown that one can create

simple, but effective, predictive probabilistic models that can make adaptation decisions on each method invocation, with an overhead ranging from a few hundred microseconds to about a millisecond, depending on a system's configuration. This suggests that computing power has increased to the point that it might be possible to use on-line models to make intelligent, autonomic, adaptation decisions in QoS-driven adaptable distributed systems. However, many issues remain before we can create such systems, including issues related to the mechanics and statistics of data collection, the extraction of appropriate model input parameter values from the collected data, the creation of models that are accurate enough to be useful but efficient enough in their solution to be solved quickly, and methods to insure the stability of the recommended sequence of adaptation. Furthermore, since a single monolithic model will not suffice to control a large distributed system, techniques need to be developed to coordinate the actions of multiple adaptation models present in a system in a coordinated way, without requiring the exchange of an unreasonable amount of state data. The AQuA project has created the theory, algorithms and implementations necessary to build dependable distributed systems, and also provided inspiration for intrusion-tolerant and model-driven quality-of service-enabling middleware.

3.10. References

[Bir96] K. P. Birman, *Building Secure and Reliable Network Applications*, Greenwich, CT: Manning Publications, 1996

[Fel96] P. Felber, B. Garbinato, and R. Guerraoui, "The Design of a CORBA Group Communication Service," *Proc. 15th IEEE Symposium on Reliable Distributed Systems*, Niagara on the Lake, Ontario, Canada, Oct. 1996, pp. 150-159

[Hay98] M. G. Hayden, "The Ensemble System," Ph.D. thesis, Cornell University, 1998

[Loy98a] J. P. Loyall, R. E. Schantz, J. A. Zinky, and D. E. Bakken, "Specifying and Measuring Quality of Service in Distributed Object Systems," *Proc. First International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'98),* Kyoto, Japan, Apr. 1998

[Loy98b] J. P. Loyall, D. E. Bakken, R. E. Schantz, J. A. Zinky, D. A. Karr, R. Vanegas, and K. R. Anderson, "QoS Aspect Languages and Their Runtime Integration," *Lecture Notes in Computer Science*, vol. 1511: *Proc. Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR98),* Pittsburgh, PA. Springer-Verlag, May 1998

[Maf95] S. Maffeis, "Run-Time Support for Object-Oriented Distributed Programming," Ph.D thesis, University of Zurich, 1995

[Maf97] S. Maffeis, "Piranha: A CORBA Tool for High Availability," *IEEE Computer*, vol. 30, no. 4, pp. 59-66, 1997

[Mor99] G. Morgan, S. K. Shrivastava, P. D. Ezhilchelvan, and M. C. Little, "Design and Implementation of a CORBA Fault-Tolerant Object Group Service," *Proc. Second IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems (DAIS'99)*, Helsinki, June 1999

[Mos98] L. E. Moser, P. M. Melliar-Smith, and P. Narasimhan, "Consistent Object Replication in the Eternal System," *Theory and Practice of Object Systems*, vol. 4, no. 2, pp. 1-12, 1998

[Nar97] P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith, "Replica Consistency of CORBA Objects in Partitionable Distributed Systems," *Distributed Systems Engineering*, vol. 4, no. 3, pp. 139-150, Sept. 1997

[Nar99b] P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith, "Using Interceptors to Enhance CORBA," *IEEE Computer,* vol. 32, no. 7, pp. 62-68, July 1999

[Nar00] P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith, "Gateway for Accessing Fault Tolerance Domain," *Middleware 2000: IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, New York, NY, April 2000, pp. 88-103

[Nar01] P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith, "State Synchronization and Recovery for Strongly Consistent Replicated CORBA Objects," *Proc. of the 2001 International Conference on Dependable Systems and Networks*, pp. 261-270

[Ren01a] Y. Ren, M. Cukier, and W. H. Sanders, "An Adaptive Algorithm for Tolerating Value Faults and Crash Failures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 2, pp. 173-192, Feb. 2001

[Ren01b] Y. Ren, "AQuA: A Framework for Providing Adaptive Fault Tolerance to Distributed Applications," Ph.D. thesis, University of Illinois at Urbana-Champaign, 2001

[Rub00] P. G. Rubel, "Passive Replication in the AQuA System," Master's Thesis, University of Illinois, 2000

[TAO] Department of Computer Science, Washington University, "Real-time CORBA with TAO (The ACE ORB)," http://www.cs.wustl.edu/~schmidt/TAO.html

[Vay98] A. Vaysburd and K. P. Birman, "The Maestro Approach to Building Reliable Interoperable Distributed Applications with Multiple Execution Styles," *Theory and Practice of Object Systems*, vol. 4, no. 2, 1998

[Zin97] J. A. Zinky, D. E. Bakken, and R. E. Schantz, "Architectural Support for Quality of Service for CORBA Objects," *Theory and Practice of Object Systems*, vol. 3, no. 1, pp. 55-73, Apr. 1997

4. Real-Time CORBA: Experimental Results and Applicability to the A/V Streaming Service

OOMWorks investigators:
Pradeep Gore, Irfan Pyarali, Yamuna Krishnamurthy

4.1. Introduction

One of the emerging off-the-shelf mechanisms for helping to control real-time behavior in a standardized way is the real-time CORBA specification. We are experimenting with an implementation of RT-CORBA based on the TAO CORBA ORB. This section consists of a description of results from experimenting with that implementation of RT-CORBA and proposal of how RT-CORBA capabilities could be incorporated directly into the A/V Streaming Service and indirectly into the UAV application to help effect and control better real time behavior. The purpose of this QuOIN experimentation was to assess the suitability of available capabilities in RT-CORBA for embedded applications such as UAV with stringent real time requirements.

4.2. Real-Time CORBA

The Real-Time CORBA specification, illustrated in Figure 4.1, provides a high level API for programmers to write distributed applications in which the priority of a "distributed thread" of execution is maintained across separate hosts with potentially different operating systems. It also provides support for explicit binding, standard synchronizers and the ability to modify transport protocol properties and thread pools as a standard.
In this experiment we show the effect of maintaining end-to-end priorities.
The experiment consists of the following participants:

- Job – A CORBA servant object that performs CPU intensive work. The amount of work depends on a load factor that is conveyed to the object per invocation as an argument.
- Periodic Task – A periodic task is a thread of execution that is associated with a Job. A Task periodically invokes the Job after a period of time specified by the user.
- Activity – An activity is a collection of Job's and Tasks hosted in a single process. An activity reads a configuration file that can be used to initialize in many ways such as a client or server.
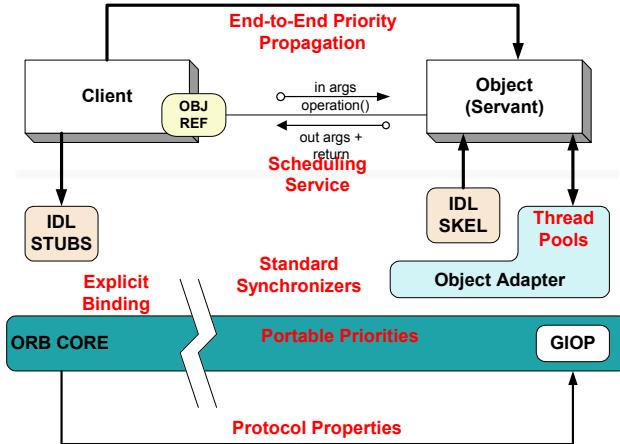
Figure 4.1: The RT CORBA Specification

4.3. Experimental setup

Figure 4.2 illustrates the setup for the experiments, with the following characteristics:

Hardware Profile
OS: Linux 2.4 (Redhat 7.1)
Processor (2): Intel Pentium III 930 MHz
Memory: 500 Megabytes
CPU Cache: 256 KB

Threads Profile
ORBSchedPolicy: SCHED_FIFO
ORBScopePolicy: SYSTEM
ORBPriorityMapping: linear

Test Bed Profile
Base Work: 30
Base Invocation Rate: 151

Priority Profile
High Priority Lane 32767
Medium Priority Lane 21844
Low Priority Lane 10922
Best Effort Lane 0
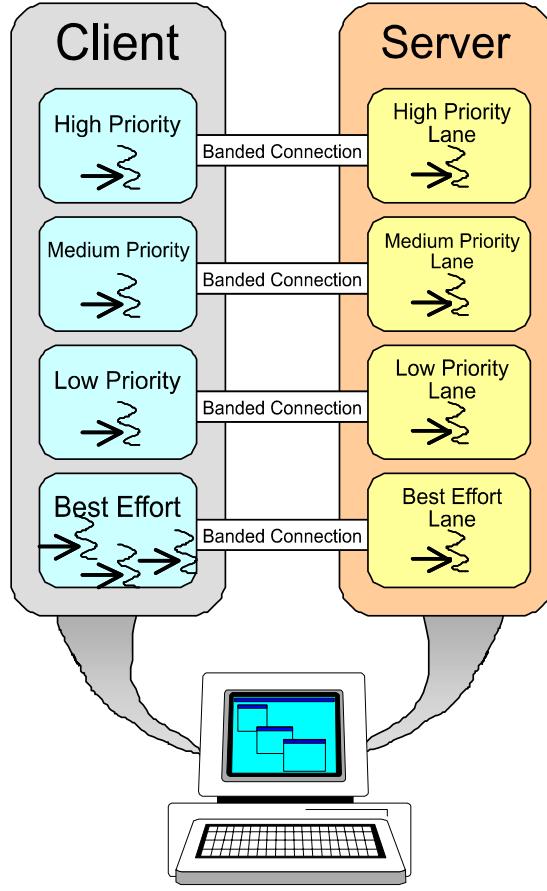
Figure 4.2: Experimental setup for the RTCORBA experiments

4.4.   Summary of results

As shown in figure 4.3(a), when the load is minimal, all tasks complete their run at the same time with the same throughput and latencies. A similar graph (not shown here) would be obtained if the threads were not assigned priorities. The thread priorities show no effect as the load is very small and hence each thread manages to get its share of the CPU.

As shown in figure 4.3(b), when the load is significantly higher (load = 1000), the highest priority task's thread gets to run the most and hence completes before the other tasks. While the highest priority task is running, the other tasks get very few chances to run and exhibit very high latencies during that time. The results in the end show the experimental runs of all the loads.

(a) Load = 0

(b) Load = 1000
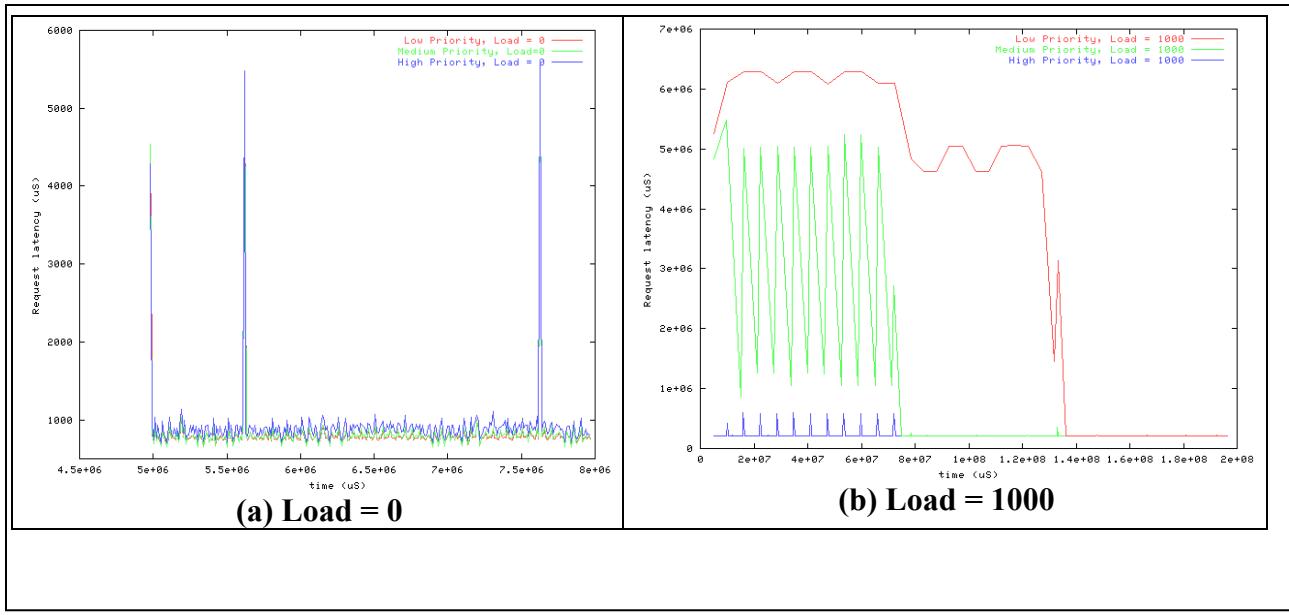
Figure 4.3: Summary of experimental results

4.5. Applying RT-CORBA concepts to A/V Streaming

The UAV Demo architecture, illustrated in Figure 4.4, simulates the streaming of video data from a UAV to a distributor host on the nearest ship. From the distributor the same data is streamed to multiple displays within the ship. In the UAV demo we have three processes- sender, distributor and receiver. The sender reads data from a file and sends it to the distributor (simulating the data streamed from the UAV). The distributor process in turn streams the data to the multiple receiver processes that display the video. The streaming between the sender and distributor, and between the distributor and host is done with the TAO AV Service.
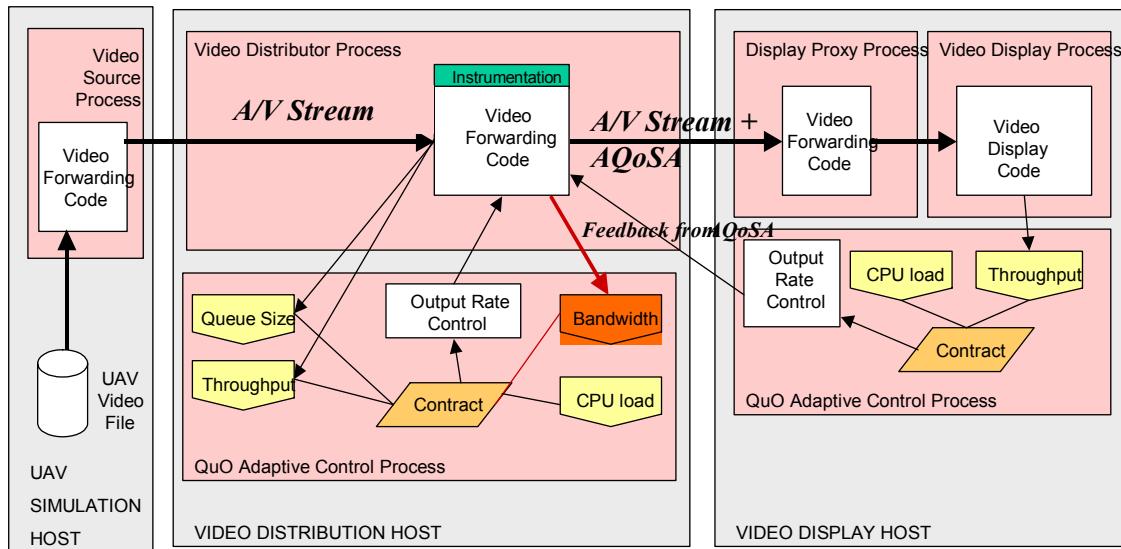
Figure 4.4: Overview of the UAV demo architecture

The TAO AV Streaming Service is an implementation of OMG's CORBA A/V Streaming Service. This service enables the control, connection and streaming of audio/video streams between multimedia devices through standardized API's. The TAO AV Service allows streaming data over different transport and flow protocols like TCP, UDP, and RTP etc. However, as it exists AV Service is currently not distinguished from any other processes that may be running on the same system. What this implies is that the AVStreams Service has to contend with other processes on the system for resources and it would be scheduled on best effort. As it exists RT-CORBA cannot be applied to differentiate the relative priorities of the multiple data streams of the AV Service. This is because the AVStreams data streaming is out-of-band.

In order to circumvent the above problem and schedule the AV Service more predictably we propose to integrate the RT-CORBA capabilities into the AV Service. Also we would like to incorporate the RT-CORBA features as demonstrated by the above experiments into the UAV demo directly.

4.5.1. Use Cases

As mentioned above we could use RT-CORBA to set priority on AV process and hence make it more predictable. As it exists now the UAV demo is able to adapt to CPU loads after the load has been detected. By incorporating RT-CORBA we could prevent such CPU loading by prioritizing the AV Service and hence preempting other processes.

Also the UAV may need to transmit other kinds of periodic data - e.g. Global Position System (GPS), Inertial Navigation Set (INS), and Forward Looking Infrared Radar. This data is represented suitably as structured information hence CORBA could be used to transfer this data. We could use RT-CORBA capabilities to prioritize these data streams. For example, if we have GPS and INS data being transmitted we could prioritize the GPS data over INS using RT-CORBA. When there is resource contention then GPS will get preferential treatment.

We could also use RT-CORBA to prioritize the receivers who are connecting to the distributor. Further, we could map RT CORBA priorities to diffserv type of service thereby ensuring true end-to-end priority preservation.

By integrating RT-CORBA with AV Service we would be able to prioritize the multiple data streams of the AV Service. This may require making the AV Service multi-threaded and dealing appropriately with the acceptors, connectors and reactors created for streaming data. If this is in place we would be able to prioritize the multiple streams entering and leaving the distributor and receiver in the UAV Demo.

4.6. Sample Results from Experiments

4.6.1. Experiment 1: Increasing Work in CORBA without RT-CORBA
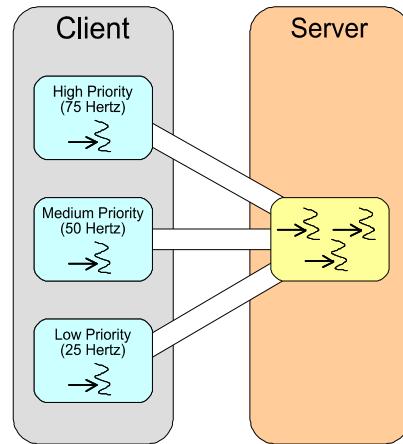
Experiment

Figure 4.5: Experiment 1: Increasing Work in CORBA without RT-CORBA Setup

- Measure the disruption caused by Increasing Work in CORBA without RT priorities
- Increasing Priority =>Increasing Rate**Server**

3 threads

Client
- 3 Rates Based Invocation threads
  - o High => 75 Hz
  - o Medium => 50 Hz
  - o Low => 25 Hz
- Work fixed at 30

Result

Conclusion
- As work increases and system capacity decreases, the high priority 75 Hertz client is effected first, followed by the medium priority 50 Hertz client, and finally by the low priority 25 Hertz client
- The above behavior is because all clients are treated equally by the server
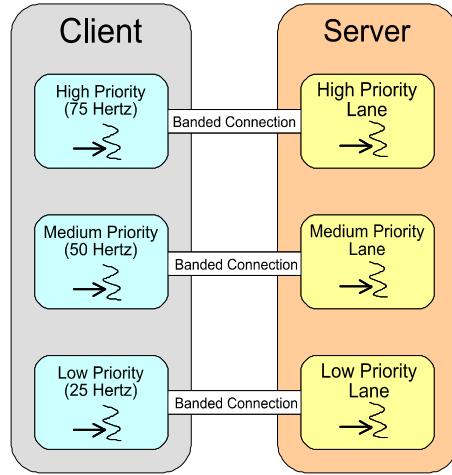
Figure 4.6: Experiment 2: Increasing Work in RT-CORBA With Lanes

4.6.2. Experiment 2: Increasing Work in RT-CORBA With Lanes (Increasing Priority => Increasing Rate)

Experiment
- Measure the disruption caused by Increasing Work in RT-CORBA With Lanes
- Increasing Priority => Increasing Rate

Server
- 3 thread lanes
- High / Medium / Low

Client
- 3 Rates Based Invocation threads
  - o High => 75 Hz
  - o Medium => 50 Hz
  - o Low => 25 Hz
- Work fixed at 30

Result

Conclusion

- As work increases and system capacity decreases, the low priority 25 Hertz client is effected first, followed by the medium priority 50 Hertz client, and finally by the high priority 75 Hertz client
- The above behavior is because higher priority clients are given preference over lower priority clients by the server

4.6.3. Experiment 3: Increasing Work in RT-CORBA With Lanes (Increasing Priority => Decreasing Rate)
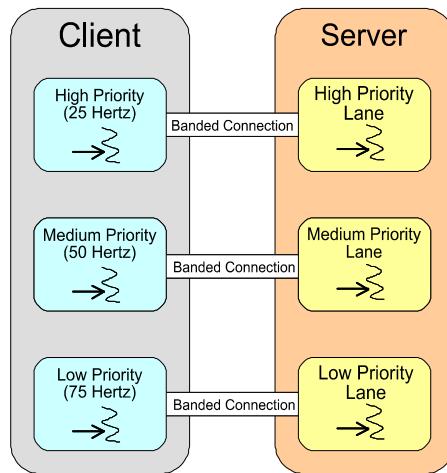
Figure 4.7: Experiment 3: Increasing Work in RT-CORBA With Lanes

Experiment

- Measure the effect of Increasing Invocation Rate in Test Bed
- Increasing Priority => Decreasing Rate

Server

- 3 thread lanes
- High / Medium / Low

Client
- 3 Rates Based Invocation threads
    - High => 75 Hz
    - Medium => 50 Hz
    - Low => 25 Hz
- Work fixed at 30

Result
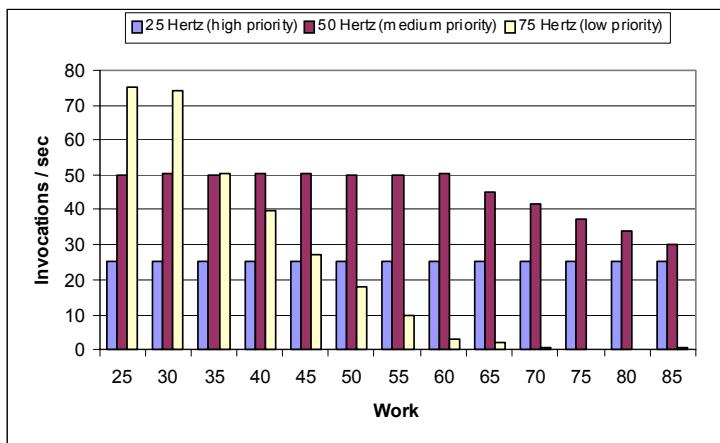
Conclusion
- As work increases and system capacity decreases, the low priority 75 Hertz client is effected first, followed by the medium priority 50 Hertz client, and finally by the high priority 25 Hertz client
- The above behavior is because higher priority clients are given preference over lower priority clients by the server

4.6.4   Experiment 4: Increasing Best Effort Work in CORBA without RT-CORBA

Experiment
- Measure the disruption caused by the Increasing Best Effort Work in vanilla CORBA
- Increasing Priority => Increasing Rate
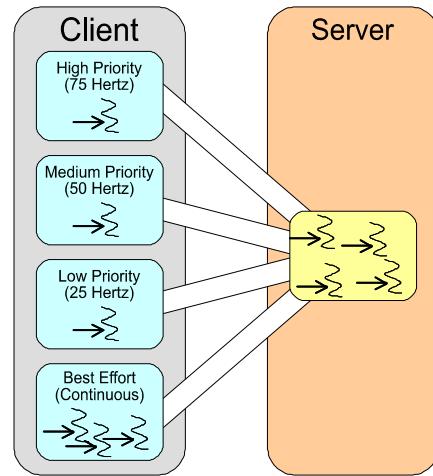
Figure 4.8: Experiment 4: Increasing Best Effort Work in CORBA without RT-CORBA

Server
- 4 threads

Client
- 3 Rates Based Invocation threads
  - High => 75 Hertz
  - Medium => 50 Hertz
  - Low => 25 Hertz
- Several Best Effort threads => Continuous Invocations

Notes
System is running at capacity => Any progress made by Best Effort threads will cause disruptions
Result
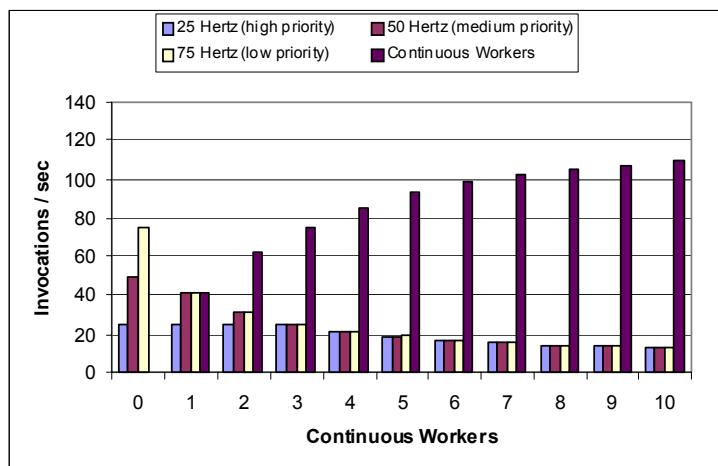
Conclusion
- All three priority based clients suffer as the number of best effort clients are added to the system

- The above behavior is because all client threads are treated equally by the server

4.6.5   Experiment 5: Increasing Best Effort Work in RT-CORBA With Lanes

Experiment
- Measure the disruption caused by Increasing Best Effort Work in RT-CORBA With Lanes
- Increasing Priority =>Increasing Rate
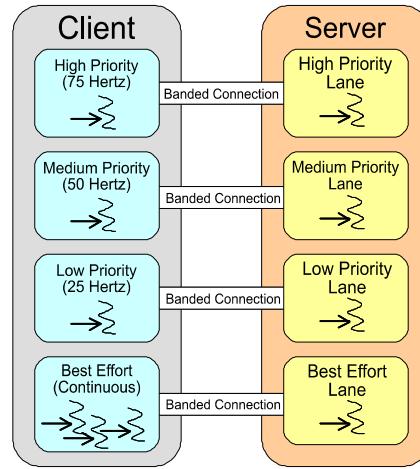
Figure 4.9: Experiment 5: Increasing Best Effort Work in RT-CORBA With Lanes

Server
- 4 thread lanes
- High / Medium / Low / Best Effort

Client
- 3 Rates Based Invocation threads
    - o   High => 75 Hertz
    - o   Medium => 50 Hertz
    - o   Low => 25 Hertz
- Several Best Effort threads => Continuous Invocations

Notes
- System is running at two levels
- At capacity => Any progress by Best Effort threads will cause disruptions
- Just below capacity => Best Effort threads should be able to capture any slack in the system

Result A: Increasing Best Effort Work in RT-CORBA With Lanes
System Running at Capacity (Work = 30)
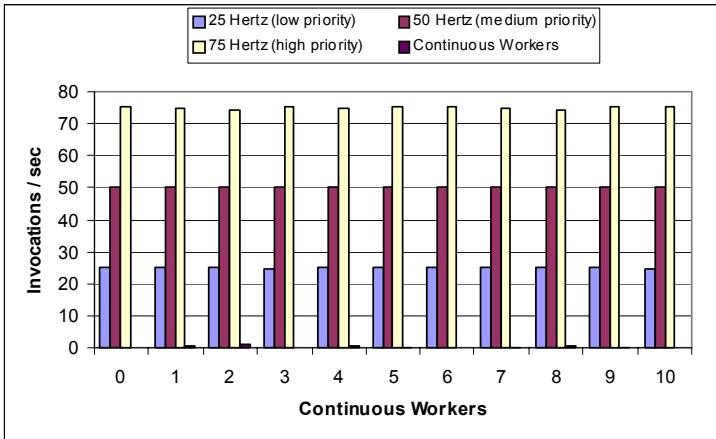
Result B: Increasing Best Effort Work in RT-CORBA With Lanes
System Running Slightly Below Capacity (Work = 28)
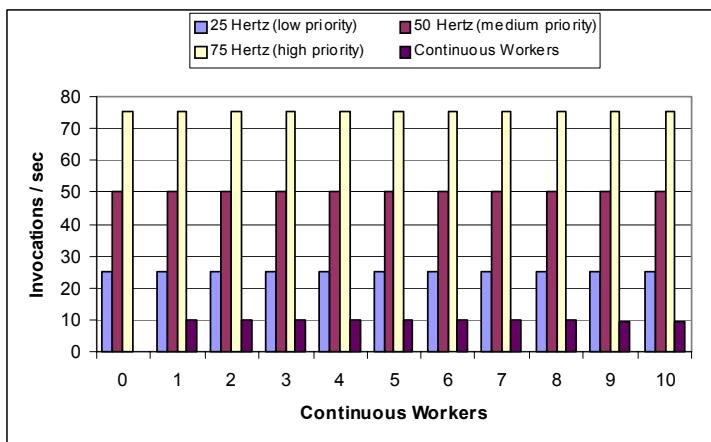
Conclusion

- Increasing Best Effort Work in RT-CORBA With Lanes
- Addition of best-effort client threads did not effect any of the three priority based clients
- Best-effort client threads were limited to picking up slack left in the system
- As the number of best-effort client threads increase, throughput per best-effort client thread decreases, but the collective best-effort client throughput remains constant

4.6.6 Experiment 6: Increasing Work in RT-CORBA Without Lanes

Experiment

- Measure the disruption caused by Increasing Work in RT-CORBA Without Lanes
- Increasing Priority => Increasing Rate
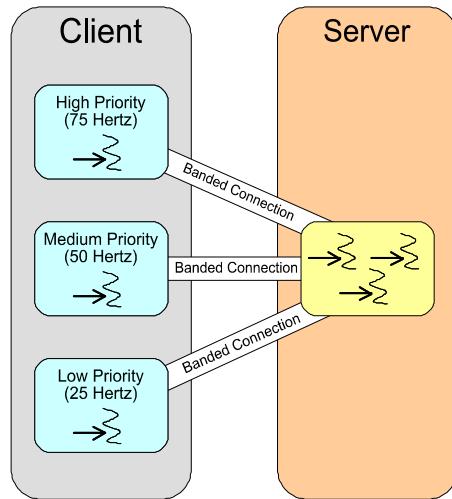
Figure 4.10: Experiment 6: Increasing Work in RT-CORBA Without Lanes

Server
- 3 threads in pool

Client
- 3 Rates Based Invocation threads
  - High => 25 Hertz
  - Medium => 50 Hertz
  - Low => 75 Hertz

Notes
- Server pool priority will be varied
Low / Medium / High
Result A: Increasing Work in RT-CORBA Without Lanes, Server Pool Priority = Low
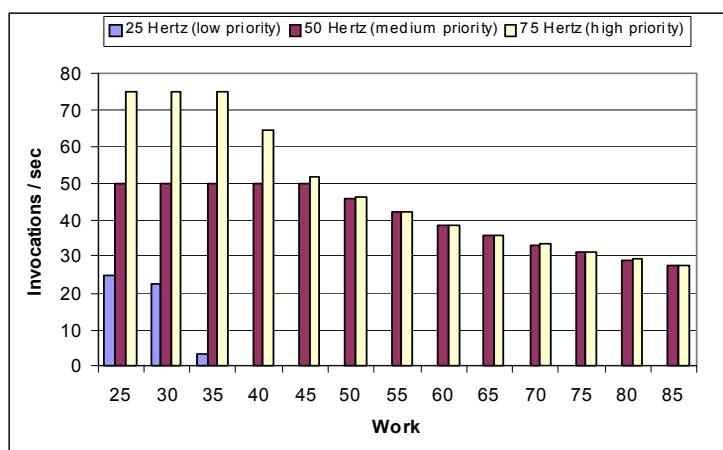
Result B: Increasing Work in RT-CORBA Without Lanes
Server Pool Priority = Medium
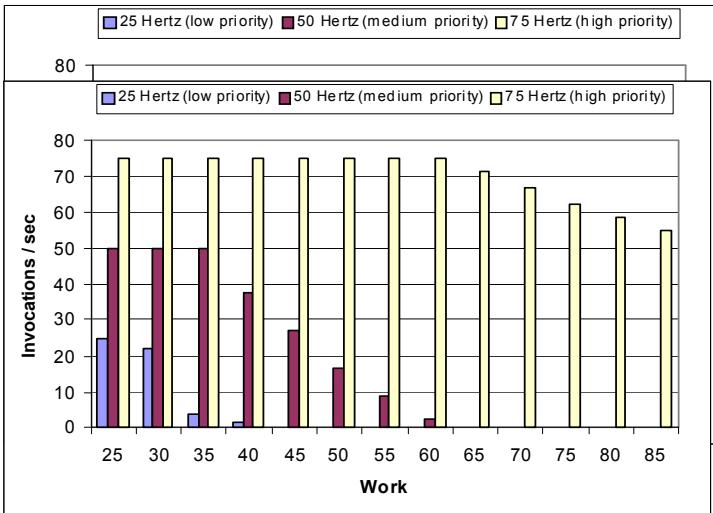
Result C: Increasing Work in RT-CORBA Without Lanes
Server Pool Priority = High

Conclusion

- When the server pool is running at low priority, a pool thread cannot preempt an upcall thread when a new request arrives from a client thread of higher priority. Therefore, all three client threads receive similar service from the server
- When the server pool is running at medium priority, a pool thread can only preempt an upcall thread running at low priority when a new request arrives from a client thread of medium or high priority. Therefore, the medium and high priority client threads receive similar service from the server

When the server pool is running at high priority, a pool thread can preempt upcall threads running at low and medium priorities when a new request arrives from a client thread of high priority. Therefore, the high priority client thread receives the best service from the server

5. Synopsis of the 3 Major QuOIN Software Releases to the Quorum Technical Community

QuO/QuOIN Software Release History

One of the major accomplishments of the QuOIN activity has been the formalized collection, release and support of integrated software packages of Quorum technology centered around the distributed object framework into the Quorum community and for transition application development. There have been three major release cycles, with the third ushering in a more dynamic strategy of daily builds and frequent minor releases in place of less frequent major releases. Beginning with the third major release cycle, most of thre available software has been made available via the open-source licensing model. The following table is a chronological release history of the QuO software and accompanying QuOIN integration components. This is followed by a brief description of the increasing components and capabilities associated with each of the three major software releases.

December 2001	QuO/QuOIN release 3.0.9
October 2001	QuO/QuOIN release 3.0.8
October 2001	QuO/QuOIN release 3.0.7
September 2001	QuO/QuOIN release 3.0.6
July 2001	QuO/QuOIN release 3.0.5
June 2001	QuO/QuOIN release 3.0.4
May 2001	QuO/QuOIN release 3.0
December 1999	QuO/QuOIN release 2.1
May 1999	QuO/QuOIN release 2.0
September 1998	QuO/QuOIN release 1.0

5.1. Release 1.0 Contents and Environment
September 1998

This QuO release assumes the following software:
ORB:
- Visigenic's VisiBroker 3.2 (CORBA 2.1; IIOP 1.0)

Languages:
- Java (JDK 1.1.5)
- C++ (GNU and Sun)

OSs:
- Solaris 2.5
- Red Hat Linux 5.0 (Hurricane), Kernel 2.0.32 on an i686 (Pentium)

This QuO release provides the following:

QuO Base:
- QuO Kernel
- QuO System Condition Object Library (initial release)
- QuO Compilers for CDL, SDL
- Sample QuO Contracts
- Preliminary Instrumentation

Mechanisms:
- RSVP Gateway
- Replication Gateway
- Proteus Dependability Manager

Note: the mechanisms are currently provided "as is" for purposes of edification and elucidation only. They will be released in two separate files (one for DIRM, one for AQuA), along with the AQuA and DIRM demos, and will be supported in later releases.

Demo Applications:
- Simple (Java and C++ client)
- Slideshow (Java; originally demonstrated at the (DARPA ITO) July '98 Quorum PI Meeting)
- AQuA Slideshow (Slideshow above, but with replication management)
- DIRM Slideshow (Slideshow above, but with bandwidth management)

Note that the AQuA and DIRM Slideshow demos are released in separate distributions from this release. They are described here, however, to help QuO users better understand what using mechanisms to "reserve" properties such as replication and communication entail.

Packaging and Documentation

QuO Toolkit 1.0 Overview as well as the following documents is provided with this release:
- **QuO's Structure Description Language (SDL)**
- **QuO's Contract Description Language (CDL)**
- **How to Write a System Condition Object**
- **How to Write a Connection Object**
- **QuO Packages**
- **QuO Kernel GUI**

5.2. Release 2.0 Contents and Environment

May 1999

QuO version 1.0, released September 1998, included the QuO runtime kernel, Contract Description Language and Structure Description Language and their code generators, system condition object library, concept demonstration gateway, example applications including a network bandwidth resource reservation example and documentation.

QuOIN Version 2.0 included:
- Improved QuO language and contract support

- A connector language for automating the hookup of delegates, contracts and system condition objects with objects and clients
- Instrumentation support including Remos
- A revised general gateway design[1]
- Quorum Distributed Object individual QoS property incremental results:
- Real time support using the TAO ORB and schedulable event channels
- Bandwidth management using QosME, RSVP & Beagle[1]
- Dependability using Proteus and Ensemble
- Adaptive caching
- Object security using OODTE ORB based access control with dynamic security policy[1]

QuOIN Release V2
Languages: Java (Visigenic, JDK), C++ (Visigenic, TAO)
OS: Solaris 2.7, Linux Redhat 5.2 and NT 3.51

Middleware:
Corba v2.2; including use of POA; Name Server
Orb: Interoperable set selected from Visigenic v3.3, TAO v1.0 (**), JDK v.1.2 Orb (some attributes may be Orb specific)
IDL compiler: Orb specific, plus FLICK emerging as common tool (currently only for TAO)

QuO V2 including:
* Multiple contract support
     - The code generators will generate nested delegates, each delegate corresponding to a contract. Revised generated QuO Runtime code to support multiple contracts

*Extended QDL including CDL, SDL and ConnDL
     - A single quogen, which works on a connection language specification, instead of separate quogen-contract and quogen-delegate
     - Connection language (CSL or ConnDL), which specifies the IDL, CDL, and SDL files involved in a QuO application, the objects that are involved in the application, and how to hook them up. The processing of this code by the quogen executable would generate the contract and delegate code, plus the connection code that was previously written by hand.
     -Improved SDL, with local variable declarations, multiple bindings, more powerful adaptation descriptions and support for instrumentation

*Instrumentation package

*Reuseable QuO Gateway shell using Corba pluggable protocols (James) improved runtime kernel including GUI, enhanced SysCon library, all bugs and deficiencies with version 1 reported to QuO help fixed

Mechanisms (***):

---

[1] Released as part of interim release 2.1; see below

*Extended Bandwidth management capability (new gateway shell with embedded RSVP, REMOS data collection, Darwin reservations, SMARTS event management, Qosme socket management)

*Real time support (TAO specialized real time ORB+RIDL support, TAO specific Real-time Scheduling Service for rate monotonic event queues and schedulers, and dynamic scheduling strategies, such as MUF, EDF and MLF)

*AQuA dependability management capability including Proteus property manager, selections of off-the-shelf handlers for customized operation and customization interfaces via new handler development
      -passive replication strategy in addition to active
      -Dynamic, run-time configuration of client-server relationships
            First level of this is that we know what services each client wants at compile time, we just don't decide on the port numbers etc. until run time (so we can for example use transitory rather than persistent IORs)

*Inter-ORB access control (using SIGMA OO-DTE interceptors)

Other accessories:
Substitutable Real Time OS for use with TAO RT ORB
Testing and validation suite: Embedded validation through region monitoring Gui, reports and logs, and through Loki fault injector testing tool

Demonstration applications including delegates, sysconds, handlers, controlling and providing adaptability for dependability, real time control, bandwidth mgmt, dynamic security properties and an interim integrated suite

Packaging with revised configuration management, and revised documentation, including concept of operation and programming examples
UML like Design spec, Quorum interfacing spec and use examples

Notes:
 (**) The TAO ORB comes with the following set of CORBA services:
All CORBA 2.2 features except for Interface Repository, plus pluggable transport protocols, multiple threading models (such as Thread Pool, Thread-per-Request, Thread-per-Connection and Reactive models), support for OMG Object Services including Naming Service, Trading Service, Property Service, Events Service, Audio/Video Streaming Service, Concurrency Service, Lifecycle Service and Time Service
(***) For this version, embedded mechanisms (RSVP, Ensemble, Proteus, Remos) may not run on NT and others (OO-DTE) may not run on Solaris.

5.3. Release 3.0 Contents and Environment

May 2001

The 3.0 release of the Quality Objects framework for distributed system development (QuO 3.0) was made available in May 2001. This release is based around the QuO toolkit, which includes the following features:

- QuO runtime kernel, including GUI visualization capability
- QuO code generators for Quality Description Languages (QDL), including the Contract Description Language
- (CDL) and the Aspect Structure Language (ASL) components
- Support for built-in and user-specified instrumentation for QuO-enhanced CORBA or RMI applications
- Support for Java and C++ applications
- Support for CORBA and Java RMI distributed object middleware
- A library of sample QuO system condition objects
- Several application and QoS property examples that demonstrate how to build applications and QoS mechanisms
- Using the QuO framework
- A gateway mechanism for hooking in special purpose transport layers below the ORB
- On-line documentation and how-to descriptions

QuO 3.0 adds the following new capabilities and features, as well as significant improvements to each of the separate modules for the QoS attribute integration activities:
- Qoskets: mechanism for bundling QoS adaptive code for reuse
- C++ integrated QuO Kernel
- Java RMI support
- Aspect SDL (ASL):
- Language independent (C++ and Java) with Java-like syntax
- Multiple ASL files can be woven together
- ASL Templates to indirect Method signatures.
- State Machine Representation for Regions in CDL
- Resource Status Service: Unified architecture for monitoring the status of external resources from inside an application
- Default resource configurations published on web pages
- QuO Status Typed Event Channel push technology for getting host load and capacity
- Direct interface to Remos, for network monitoring
- Reusable in-band instrumentation, i.e., an instrumentation Qosket allows instrumentation independent of the Business interface
- Example Adaptive Code for Resource Monitoring with Client-side adaption (RSS and in-band instr)
- Bandwidth Reservation (RSVP)
- Security (Authentication)
- Easy to install Linux RPMs

The QuO release consists of the following components:
- QuoCore
    - The QuO base system, including the codegenerator and runtime kernels
- quoDoc
    - Written documentation including the Users' and Reference Guide Doxygen and Javadoc generated documentation for examples

- quoOODTE
    - -Security service based on OODTE from NAI
- quoRemos
    - -Resource management service based on Remos
- quoRsvporb
    - -Bandwidth reservation service based on Rsvp
- quoAqua
    - -Replication Service based on AQuA
- quoUAV
    - -Unmanned Airborne Vehicle (UAV) distributed QoS video application

Supported Platforms for QuO V.3:
The QuO v.3 QDL language processors and code generators are distributed in binaries for RedHat 6.2, RedHat 7.1, SunOS 5.6 and Windows NT 4.0. They generate Java and C++ code for CORBA and Java code for Java RMI. They support CORBA 2.4, specifically Visibroker 4.51, JacORB 1.3.30 for Java and TAO 1.1 for C++.

QuO software is now open-source. The QuO software is released under an open-source license. By downloading the software you agree to the terms of this license.

5.4. AQuA Dependability Module Implementation Status

The current implementation of AQuA is documented with its own User's/Programmer's Manual. The current version runs on both Unix (Solaris SPARC and Intel x86) and Windows 2000 (Intel x86) operating systems, and with VisiBroker, TAO, and JacORB ORBs. A typical configuration would include multiple machines connected via an Ethernet.
The required infrastructure for this version of AQuA is:
Solaris 2.7 for Sun Sparc, RedHat Linux 6.2, Windows NT/2000
ACE - TAO 5.2.1
Ensemble version 1.20-AQuA (provided)
VisiBroker for Java 4.5.1
JacORB 1.4
Java version 1.2.2 or Java version 1.3
gcc version 2.95.2

For more information, see the User's/Programmer's Manual or the AQuA website at www.crhc.uiuc.edu/PERFORM/AQuA.html.

6. An R&D Plan for Moving Forward

In this section we speculate on where these types of adaptive QoS middleware activities are or ought to be headed, based on experience gained from the Quorum and QuOIN activities. In doing so, we take a broader view of the challenges we face in constructing the new generation of Distributed Real-Time Embedded (DRE) systems, going well beyond the relatively simple examples developed to date in Quorum. In our view, the key research challenges for the next several years will involve the integration and augmentation of the following capabilities:

- *Contracts and adaptive meta-programming* – Information must be gathered for particular applications or application families regarding user requirements, resource requirements and system conditions. Multiple system behaviors must be made available based on what is best under the various conditions. This information provides the basis for the contracts between users and the underlying system substrate. These contracts provide not only the means to specify the degree of assurance of a certain level of service, but also provide a well defined, high-level middleware abstraction to improve the visibility of adaptive changes in the mandated behavior. A crucial (and currently missing) capability involves the ability to change strategies rapidly and with few negative side effects.

- *Graceful degradation* – Adaptive meta-programming mechanisms must also be devised to monitor the system and enforce contracts, providing feedback loops so that application services can degrade gracefully (or augment) as conditions change, according to a prearranged contract governing that activity. The initial challenge here is to establish the idea in developers' and users' minds that multiple behaviors are both feasible and desirable. The next step is to put into place the additional middleware support–including connecting to lower level network and operating system enforcement mechanisms–necessary to provide the right behavior effectively and efficiently given current system conditions.

- *Prioritization and physical world constrained load invariant performance* – Some systems are highly correlated with physical constraints and have little flexibility in some of their requirements for computing assets, including QoS. Deviation from requirements beyond a narrowly defined error tolerance can sometimes result in catastrophic failure of the system. The challenge is in meeting these *invariants* under varying load conditions. This often means guaranteeing access to some resources, while other resources may need to be diverted to insure proper operation. Generally collections of such components will need to be resource managed from a system (aggregate) perspective in addition to a component (individual) perspective.

Although it is possible to satisfy contracts, achieve graceful degradation and globally manage some system resources to a limited degree in a limited range of systems today, much R&D work remains. The research strategies needed to deliver these goals can be divided into the seven areas described below.

1. Individual QoS Requirements – Individual QoS deals with developing the mechanisms relating to the end-to-end QoS needs from the perspective of a single user or DRE application. The specification requirements include multiple contracts, negotiation and domain specificity. Multiple contracts are needed to handle requirements that change over time and to associate several contracts with a single perspective, each governing a portion of an activity. Different users running the same application may have different QoS requirements emphasizing different benefits and tradeoffs, often depending on current configuration. Even the same user running the same application at different times may have

different QoS requirements, *e.g.,* depending on current mode of operation and other external factors. Such dynamic behavior must be taken into account and introduced seamlessly into next-generation distributed systems.

General negotiation capabilities that offer convenient mechanisms to enter into and control a negotiated behavior (as contrasted with the service being negotiated) need to be available as COTS middleware packages. The most effective way for such negotiation-based adaptation mechanisms to become an integral part of QoS is for them to be "user friendly," *e.g.,* requiring a user or administrator to simply provide a list of preferences. This is an area that is likely to become domain-specific and even user-specific. Other challenges that must be addressed as part of delivering QoS to individual applications include:

- Translation of requests for service among and between the various entities on the distributed end-to-end path,
- Managing the definition and selection of appropriate application functionality and system resource tradeoffs within a "fuzzy" environment and
- Maintaining the appropriate behavior under composability.

Translation addresses the fact that complex network-centric systems are being built in layers. At various levels in a layered architecture the user-oriented QoS must be translated into requests for other resources at a lower level. The challenge is how to accomplish this translation from user requirements to system services. A logical place to begin is at the application/middleware boundary, which closely relates to the problem of matching application resources to appropriate distributed system resources. As system resources change in significant ways, either due to anomalies or load, tradeoffs between QoS attributes (such as timeliness, precision and accuracy) may need to be (re)-evaluated to ensure an effective level of QoS, given the circumstances. Mechanisms need to be developed to identify and perform these tradeoffs at the appropriate time. Last, but certainly not least, a theory of effectively composing systems from individual components in a way that maintains application-centric end-to-end properties needs to be developed, along with efficient implementable realizations of the theory.

2. Run-time Requirements – From a system lifecycle perspective, decisions for managing QoS are made at design time, at configuration/deployment time and/or at run-time. Of these, the run-time requirements are the most challenging since they have the shortest time scales for decision-making, and collectively we have the least experience with developing appropriate solutions. They are also the areas most closely related to advanced middleware concepts. This area of research addresses the need for run-time monitoring, feedback and transition mechanisms to change application and system behavior, *e.g.,* through dynamic reconfiguration, orchestrating degraded behavior or even off-line recompilation. The primary requirements here are *measurement*, *reporting*, *control*, *feedback* and *stability*. Each of these plays a significant role in delivering end-to-end QoS, not only for an individual application, but also for an aggregate system. A key part of a run-time environment centers on a permanent and highly tunable measurement and resource status service as a common middleware service, oriented to various granularities for different time epochs and with abstractions and aggregations appropriate to its use for run-time adaptation.

In addition to providing the capabilities for enabling graceful degradation, these same underlying mechanisms also hold the promise to provide flexibility that supports a variety of possible behaviors, without changing the basic implementation structure of applications. This reflective flexibility

diminishes the importance of many initial design decisions by offering late- and run-time-binding options to accommodate actual operating environments at the time of deployment, instead of only anticipated operating environments at design time. In addition, it anticipates changes in these bindings to accommodate new behavior.

3. Aggregate Requirements – This area of research deals with the system view of collecting necessary information over the set of resources across the system, and providing resource management mechanisms and policies that are aligned with the goals of the system as a whole. While middleware itself cannot manage system-level resources directly (except through interfaces provided by lower level resource management and enforcement mechanisms), it can provide the coordinating mechanisms and policies that drive the individual resource managers into domain-wide coherence. With regards to such resource management, policies need to be in place to guide the decision-making process and the mechanisms to carry out these policy decisions.

Areas of particular R&D interest include:
- *Reservations*, which allow resources to be reserved to assure certain levels of service
- *Admission control mechanisms*, which allow or reject certain users access to system resources
- *Enforcement mechanisms* with appropriate scale, granularity and performance and
- *Coordinated strategies and policies* to allocate distributed resources that optimize various properties.

Moreover, policy decisions need to be made to allow for varying levels of QoS, including whether each application receives guaranteed, best-effort, conditional or statistical levels of service. Managing property composition is essential for delivering individual QoS for component based applications, and is of even greater concern in the aggregate case, particularly in the form of layered resource management within and across domains.

4. Integration Requirements – Integration requirements address the need to develop interfaces with key building blocks for system construction, including the OS, network management, security and data management. Many of these areas have partial QoS solutions underway from their individual perspectives. The problem today is that these partial results must be integrated into a common interface so that users and application developers can tap into each, identify which viewpoint will be dominant under which conditions and support the tradeoff management across boundaries to get the right mix of attributes. Currently, object-oriented tools working with distributed object computing middleware provide end-to-end syntactic interoperation, and relatively seamless linkage across the networks and subsystems. There is no *managed* QoS, however, making these tools and middleware useful only for resource rich, best-effort environments.

To meet varying requirements for integrated behavior, advanced tools and mechanisms are needed that permit requests for *different* levels of attributes with different tradeoffs governing this interoperation. The system would then either provide the requested end-to-end QoS, reconfigure to provide it or indicate the inability to deliver that level of service, perhaps offering to support an alternative QoS, or triggering application-level adaptation. For all of this to work together properly, multiple dimensions of the QoS requests must be understood within a common framework to translate and communicate those requests and services at each relevant interface. Advanced integration middleware provides this common framework to enable the right mix of underlying capabilities.

5. Adaptivity Requirements – Many of the advanced capabilities in next-generation DRE system environments will require adaptive behavior to meet user expectations and smooth the imbalances between demands and changing environments. Adaptive behavior can be enabled through the appropriate organization and interoperation of the capabilities of the previous four areas. There are two fundamental types of adaptation required:

1. Changes beneath the applications to continue to meet the required service levels despite changes in resource availability and
2. Changes at the application level to either react to currently available levels of service or request new ones under changed circumstances.

In both instances, the system must determine if it needs to (or can) reallocate resources or change strategies to achieve the desired QoS. Applications need to be built in such a way that they can change their QoS demands as the conditions under which they operate change. Mechanisms for reconfiguration need to be put into place to implement new levels of QoS as required, mindful of both the individual and the aggregate points of view, and the conflicts that they may represent. In particular, fundamental concern in performing adaptive resource management is to reduce overhead and improve predictability of adaptation. If too much of the resource budget is spent monitoring and performing adaptive functions, therefore, adaptation can prove ineffective or even detrimental to system performance.

Part of the effort required to achieve these goals involves continuously gathering and instantaneously analyzing pertinent resource information collected as mentioned above. A complementary part is providing the algorithms and control mechanisms needed to deal with rapidly changing demands and resource availability profiles and configuring these mechanisms with varying service strategies and policies tuned for different environments. Ideally, such changes can be dynamic and flexible in handling a wide range of conditions, occur intelligently in an automated manner and can handle complex issues arising from composition of adaptable components. Coordinating the tools and methodologies for these capabilities into an effective adaptive middleware should be a high R&D priority.

6. System Engineering Methodologies and Tools – Advanced middleware by itself will not deliver the capabilities envisioned for next-generation embedded environments. We must also advance the state of the system engineering discipline and tools that come with these advanced environments used to build complex distributed computing systems. This area of research specifically addresses the immediate need for system engineering approaches and tools to augment advanced middleware solutions. These include:

- *View-oriented or aspect-oriented programming techniques,* to support the isolation (for specialization and focus) and the composition (to mesh the isolates into a whole) of different projections or views of the properties the system must have. The ability to isolate, and subsequently integrate, the implementation of different, interacting features will be needed to support adapting to changing requirements.
- *Design time tools and models*, to assist system developers in understanding their designs, in an effort to avoid costly changes after systems are already in place (this is partially obviated by the late binding for some QoS decisions referenced earlier).

- *Interactive tuning tools*, to overcome the challenges associated with the need for individual pieces of the system to work together in a seamless manner.
- *Composability tools*, to analyze resulting QoS from combining two or more individual components.
- *Modeling tools for developing system performance models* as adjunct means (both online and offline) to monitor and understand resource management, in order to reduce the costs associated with trial and error.
- *Debugging tools*, to address inevitable problems.

7. Reliability, Trust, Validation, and Certifiability – The dynamically changing behaviors we envision for next-generation large-scale, network-centric systems are quite different from what we currently build, use and have gained some degrees of confidence in. In particular, since adaptive meta-programming mechanisms must monitor and respond to externally induced stimuli, they are vulnerable to malicious behavior. Considerable effort must therefore be focused on assuring the validity of inputs and administrative operations, validating the correct functioning of the adaptive behavior and on understanding the properties of large-scale systems that try to change their behavior according to their own assessment of current conditions, before they can be deployed. But even before that, longstanding issues of adequate reliability and trust factored into our methodologies and designs using off-the-shelf components have not reached full maturity and common usage, and must therefore continue to improve. The current strategies organized around anticipation of long life cycles with minimal change and exhaustive test case analysis are clearly inadequate for next-generation dynamic systems with stringent QoS requirements.

7. Chronological Review of QuOIN Activities

Chronological Synopsis of QuOIN activities by month, since inception:

June 1998

During June 1998, effort was devoted primarily to contract startup activities, and to preparation for the Quorum Principal Investigators' meeting in July 1998. BBN and WUSTL intend to use the PI meeting as an opportunity to acquaint Quorum researchers with recent developments in BBN's Quality Objects (QuO) research projects and to conduct an informal kickoff of QuOIN activities with the Quorum research community.

We completed definition of QuO demonstrations to be conducted at the Quorum PI meeting in San Diego. One basic demonstration application, the "Bette Davis Slideshow," will be used to demonstrate the basics of the QuO development environment, as well as specific QuO mechanisms developed under the AQuA and DIRM projects.

We selected the demonstration and integration environment. Basic components are Linux (RedHat 5.0), Java (JDK 1.1.5) and CORBA (VisiBroker 3.0).

We began working with DARPA and other Quorum researchers to define the charter of a Quorum distributed object working group. This group will provide technical input for ongoing QuOIN efforts and will provide an opportunity for Quorum researchers to participate in QuOIN efforts.
BBN participated in preliminary meetings with the Quorum integration contractor team (Teknowledge and The Open Group [TOG]). Discussion topics included arriving at common integration goals and rationalization of schedules between the QuOIN and QUITE teams. We also worked with representatives of TOG and Honeywell to support TOG's Quorum Integration Jumpstart integration demonstration.

July 1998

During July 1998, our greatest efforts were devoted to activities surrounding the Quorum PI meeting in San Diego. BBN and WUSTL made presentations, led and participated in working groups and participated in discussions in support of the QuOIN effort. We made significant progress toward defining goals and recruiting research partners for the next year's Quorum distributed object integration effort.

We completed implementation and conducted demonstrations of QuO software at the Quorum PI meeting in San Diego. These demonstrations highlighted developments under QuOIN and related projects, including the AQuA, DIRM and Open Implementation Toolkit projects. Capabilities demonstrated included the following:

- The QuO development environment, including runtime kernel and QDL code generators
- Ability to augment a standard CORBA application (the "Bette Davis Slideshow") with different QoS properties, including basic implementations of managed network bandwidth (via RSVP, based on DIRM project research) and object replication (via Ensemble/Proteus, based on AQuA project research)

- Instrumentation for measuring performance of DOC software running under QuO

At the Quorum PI meeting, we led two working groups on Distributed Objects and QoS Specification, respectively. We used the Distributed Objects working group as an unofficial kickoff venue for the QuOIN effort. We identified several key integration partners and established initial technical agendas for each of the key QoS properties to be investigated under QuOIN.

August 1998

August 1998 was chiefly devoted to project technical planning and coordination. We conducted a project kickoff meeting and reached agreement on an initial project technical plan. We hosted a formal project kickoff meeting at BBN Technologies, Cambridge on 12 August 1998. We refined the QuOIN project plan and reviewed it with DARPA and AFRL customer representatives at the project kickoff meeting 12 August 1998. Highlights of the plan include the following:
- Identification of QoS dimensions to be addressed in QuOIN project (real-time behavior, security via access control, bandwidth management, availability) and candidate software mechanisms to be integrated
- Semi-annual releases of integrated QuO software incorporating incremental additional QoS capabilities, beginning with QuO release 1.0 in September 1998
- Semi-annual "release preview demonstrations" nominally on a staggered schedule between software releases

We participated in ongoing meetings and discussions with the Quorum lead integration team. We discussed QuOIN and QUITE schedules and expected capabilities for use in Quorum integration products.

We began identification of steps towards integration of QuO and TAO, for support of real-time behavior in QuOIN. Initial phases focus on modifying TAO, with the intent of porting QuO to TAO in the next few months. Specific TAO tasks addressed during this period include the following:
- We are enhancing the TAO IDL compiler to generate code that supports either CORBA-style or native C++ exceptions.
- We began enhancements of the TAO portable object adapter (POA) to support the Dynamic Skeleton Interface. This feature will be useful in implementing the QuO gateway.
- We are enhancing the real-time components and applications in TAO to visualize the behavior of a variety of real-time scheduling algorithms, including rate monotonic scheduling (RMS), earliest deadline first (EDF), minimal laxity first (MLF) and maximal urgency first (MUF).

We continued working towards planned September 1998 initial release of QuO software to Quorum community.

We attended and participated in a DARPA/Microsoft workshop on the use of Windows NT in research projects.

September 1998

The highlight of September 1998 was the first release of QuO software, QuO version 1.0.

We completed the release of QuO version 1.0. The release was coordinated with a simultaneous release of software from the Dynamic Integrated Resource Management (DIRM) project, which provides an initial managed bandwidth capability for QuO. Key elements of QuO version 1.0 include the following:

- QuO runtime kernel, including GUI visualization capability
- QuO code generators for Quality Description Language (QDL), including Contract Definition Language (CDL) and Structure Definition Language (SDL) components
- Built-in instrumentation for QuO-enhanced CORBA applications
- Support for Java and C++ applications
- Library of sample QuO system condition objects
- Several application and QoS property examples that demonstrate how to build applications and QoS mechanisms using the QuO framework
- On-line documentation and how-to descriptions

We continued our interactions with the Quorum lead integration team. Items of interest include the following:

- We submitted a QUITE component nomination form for QuOIN to the integration team
- We hosted several informal meetings with the QUITE team's designated technical representative for QuOIN technology (Bill LaForge of The Open Group). We introduced him to QuO concepts, familiarized him with QuO plans and discussed possible applications
- We participated in discussions regarding QUITE validation

We continued design and development efforts towards integration of QuO and TAO, for support of real-time behavior in QuOIN. Specific TAO tasks addressed during this period include the following:

- We continued work on enhancing the TAO IDL compiler to generate code that supports either CORBA-style or native C++ exceptions.
- We began enhancements of the protocol engine to support dynamic *any*s, which will be useful in implementation of incremental demarshaling in the QuO gateway.
- We completed enhancements of the TAO portable object adapter (POA) to support the Dynamic Skeleton Interface. This feature will be useful in implementing the QuO gateway.
- We continued work on enhancing the real-time components and applications in TAO to visualize the behavior of a variety of real-time scheduling algorithms.

October 1998

Our primary focus during October 1998 was the exploration of techniques for implementing TAO-based versions of various components of the QuO framework.

We identified key research threads for the next version of the QuO framework. In general these threads address issues related to integration with TAO and/or generalizing QuO interfaces to facilitate integration with a larger set of QuO mechanism developers. Important areas identified for investigation include the following:

- Naming: Current CORBA naming facilities are not adequate for describing QoS attributes of remote objects or communication channels, nor do they work well for naming some types of QuO objects, such as the replicated objects used in the AQuA availability software.

- QuO Gateway: There are several TAO capabilities that would substantially reduce the effort required to integrate new transport mechanisms with the QuO gateway.

We delivered QuO version 1.0 to the Quorum lead integration team and worked with them to assist them in familiarization with it. We worked closely with the integration team's QuOIN technical representative to walk through the components of the release, including documentation and sample applications. We discussed desirable technical attributes of applications for this winter's planned QUITE demonstration that will make best use of QuO and the AQuA availability capabilities.

We continued design and development efforts towards integration of QuO and TAO, for support of real-time behavior in QuOIN. Specific TAO tasks addressed during this period include the following:
- We completed enhancements of the TAO IDL compiler to generate code that supports either CORBA-style or native C++ exceptions. This feature is necessary to support the existing QuO 1.0 code base.
- We completed enhancements of the protocol engine to support dynamic *any*s, which will be useful in implementation of incremental demarshaling in the QuO gateway.
- We held a multi-day internal QuO/TAO cross-fertilization meeting in St. Louis where we devised specific strategies for incorporating TAO features into QuO capabilities.
- We continued work on enhancing the real-time components and applications in TAO to visualize the behavior of a variety of real-time scheduling algorithms.

We had brief discussions with Trusted Information Systems to begin developing a plan for integration of access control specification with QuO.

We made QuO version 1.0 available to several other Quorum researchers who are investigating the use of QuO in their research.

November 1998
Our primary focus during November 1998 was coordinating with additional likely research partners to establish integration goals and schedules.

We continued identification of modifications to the QuO framework that will be required for integration progress. We identified deficiencies in the current QuO implementation that will need to be corrected for integration with The ACE ORB (TAO). We identified additional capabilities needed in QuO to support extended instrumentation capabilities planned for delivery as part the next release of the QuO framework.

We completed a brief investigation of Jewel for potential use in conjunction with QuO instrumentation. We decided that Jewel integration is more appropriate at the overall Quorum project level and put this activity on hold, pending any possible QUITE team action.

Representatives of BBN and Washington University visited the University of Utah and met with the Flick research team to identify potential integration approaches. We identified the following short-term and long-term goals:
- Integrate Flick with TAO as the preferred code generator for C++ stubs and skeletons.

- Evaluate performance improvements resulting from use of Flick's optimized code generation.
- To the extent possible, move towards the use of Flick as a common IDL compiler throughout QuOIN. Given the current state of Flick development (particularly language support), exclusive use of Flick will not be achieved soon; the goal is to maximize appropriate use.

A representative of BBN visited the University of Oregon to discuss possible collaborative efforts. Researchers at the University of Oregon plan to investigate the integration of QuO into their ongoing ASSERT effort. By automatically generating QoS specifications, using QuO's Quality Description Language (QDL) as the output language and ASSERT specifications as input, U. Oregon hopes to develop the capability to use ASSERT's capabilities to perform formal reasoning on QuO contracts.

December 1998

Our primary focus during December 1998 was to continue several of the early integration activities we have recently initiated with other Quorum researchers.

We conducted initial experiments to verify operation of the QuO runtime under the Windows NT operating system. Initial experiments covering exclusively the QuO runtime components were successful. We tested the QuO runtime by building the basic Bette Davis slideshow application under Linux and executing the resulting program in NT-only and mixed NT/Linux environments. The experiments verified that the basic QuO runtime components (written in Java) operate correctly without modification in the Windows NT environment. Future experiments will be required to determine the extent of modifications that may be required to the QuO development toolkit (e.g. code generators) to support development of QuO applications under NT.

We held an internal QuOIN security integration kickoff meeting in Cambridge with Franklin Webber. Through Mr. Webber, we continued our review of TIS's SIGMA research, with an eye towards identifying potential integration approaches.

We provided as-needed support to the Quorum integration team, as they prepared for an initial QUITE system demonstration. QuOIN-delivered components demonstrated include the core QuO capability and the AQuA availability mechanism. We reviewed QUITE demonstration plans and offered suggested modifications to make better use of QuOIN-provided components, particularly the AQuA availability capability.

We began planning, both internally and jointly with representatives of Teknowledge and The Open Group, to maximize the effectiveness of our participation in the Quorum/HCC PI meeting planned for February 1999 in Atlanta.

We continued our discussions with ASSERT researchers at the University of Oregon, investigating possible approaches for translating their SCR descriptions into QuO's CDL.

We began discussions with Quorum researchers at Carnegie-Mellon University about possible integration of QuOIN with CMU's ongoing Darwin and Remos projects.

January 1999

Our primary focus during January 1999 was to prepare for the upcoming Quorum/HCC Principal Investigators' meeting. At this meeting, we participated through presentations, working groups and a software demonstration.

Representatives of BBN Technologies visited Carnegie-Mellon University (CMU) to discuss possible integration activities involving QuOIN, Darwin and Remos. The following integration activities were initiated:
- Develop a QuO reserved network bandwidth capability based on Darwin reservations, and integrate this capability with ongoing research at CMU and BBN facilities
- Develop QuO system conditions that provide an interface to system load measurements and predictions (CPU load and network bandwidth) and integrate these measurements with example QuO applications

We began in earnest preparation for the Quorum/High-Performance Computing Principal Investigators' meeting, scheduled for 17-18 February 1999 in Atlanta. We planned the QuO and QuOIN demonstration capabilities to be presented at the PI meeting. These capabilities will include the following:
- Real-time event delivery through TAO event channels
- Support for caching as an alternate availability mechanism, using software developed at Georgia Institute of Technology
- The ability to trade off real-time performance for other system QoS properties (e.g. caching timeliness), under control of a QuO contract and delegate
- Automated "bottleneck" detection and response, based on QuO instrumentation capabilities and AQuA's RSVP managed network bandwidth mechanism

We worked with DARPA and the QUITE team to refine the agenda for the PI meeting. BBN/WUSTL plan to lead a working group covering the distributed objects research area. We supported the QUITE team in defining their demonstration capability. The QUITE team is planning a live or videotaped demonstration at the PI meeting, which will include demonstrations of QuO and AQuA availability software.

February 1999
We attended the Quorum/High-Performance Computing Principal Investigators' meeting in Atlanta, 17-18 February 1999. In advance of the meeting, we conducted e-mail discussions with DARPA and QUITE representatives to help plan and coordinate working group and presentation material. At the meeting, we made presentations on QuO and QuOIN, demonstrated the QuOIN software and led a working group on distributed object integration.

We worked with researchers at the University of Utah to complete an initial integration of TAO with the Flick code generator. This capability uses Flick to generate C++ stubs for TAO. Because Flick stubs are highly optimized, we expect to observe significant performance improvements when we complete performance benchmarks.

We worked with researchers at CMU to complete a version of the QuO "bottleneck" example, using Darwin reservations as the mechanism for reserving network bandwidth. The bottleneck example is a

variation on the "Bette Davis Slideshow," in which instrumentation is used to diagnose whether slow responses are due to delays in the network or in the remote server. If network delays are identified, the application attempts to correct the problem by reserving additional network resources. In performance tests at CMU, a full-blast UDP stream was used as competing network traffic. Without reservations, the bottleneck application made no measurable progress. With reservations, the image flow was unaffected by the UDP flow.

We also worked with researchers at CMU to complete an initial integration of QuO with Remos' status collection capability, but using a non-standard Remos API. The integrated product is a set of QuO system conditions that exports Remos' estimates of expected CPU load and measured network bandwidth.

We got an early start on our Technology Transition activities, scheduled to start in June 1999, during the reporting period. We delivered the QuO/QuOIN software to The Open Group, members of the Quorum Integration (Quite) team and worked closely with them to get it running in the Quite testbed.

In another Technology Transition activity, we helped University of Oregon researchers integrate QuO and ASSERT, their formal specification language. They integrated QuO into their testbed and began work on generating QDL from ASSERT specifications, from which they can prove safety and liveness properties.

March 1999
During March, we planned the upcoming QuO v2.0 release, scheduled for May 1999, and identified the QuO capabilities to be included. Important new features include the following:
- Initial connection language (CSL), which enables QuO to automatically build "bind-time" connections based on CSL specifications, without the need for application developers to hand-code connection routines
- An instrumentation package, for non-intrusively instrumenting the method call/return path and transparently passing information along with the method call/return
- An improved QuO gateway, with instantiations for real-time (using TAO), group communication (using Ensemble) and resource reservation (using RSVP and Darwin)

We identified the necessary hardware and software environment required to perform QuO integration with CMU's Remos and Darwin efforts at BBN. We have obtained portions of the required software from CMU and begun integration at BBN.

We also followed up on the Distributed Objects Working Group we led in Atlanta last month, distributing summaries to all participants and inviting their participation in future activities.

We coordinated with The Open Group (TOG) to provide access to the QuO/TAO integration software for installation in the Quite testbed. We also met with TOG to plan support for instrumentation in QuO/QuOIN and to plan the upcoming QuOIN release.

In a move to transition some of the QuOIN efforts to standards efforts, we wrote and submitted a paper to a workshop on XML.

April 1999

We continued development of QuO functionality including bug fixes and updates to the QuO kernel and the code generators, and improving the functionality of the initial gateway prototype. We also worked on various pieces of the QuOIN functionality, including integrating the TAO real-time ORB and developing an example application using TAO real-time features under QuO control.

We also integrated OODTE access control into QuO. OODTE is technology developed at Network Associates, Inc. (NAI) and provides authentication and access control at the level of CORBA methods. Its integration into QuO means that QuO applications and the QuO kernel can now prohibit unauthorized use of their methods and can define authorized use according to a security policy. This provides a basic form of security for QuO applications.

We released QuO v.1.2.5 to The Open Group (TOG), member of the Quorum Integration (Quite) team. We also worked closely with TOG to help them develop a demo of the AQuA availability software in the Quite testbed. We also attended a meeting with STD/C and the Quite team.

May 1999

During May, we received Darwin routers and installed them in the QuO testbed. We also setup CMU's Remos and Smarts InCharge in the QuO testbed. We created demos using QuO instrumentation to automatically detect network capacity and using Columbia's RSVP ORB and CMU's Remos.

We also improved our real-time demo, which illustrated the integration of the TAO ORB's real-time event channels with the QuO framework. We developed a real-time setup language, which simplified the application's setup of real-time event channels, event suppliers and event consumers, and generated the TAO code to implement it.

We also developed an air traffic control (ATC) example using the integrated QuO/OODTE tools. The example shows the ATC application using access control and modifying its behavior in response to security-relevant changes to its environment.

We helped The Open Group (TOG), member of the Quorum Integration (Quite) team, investigate integrating AQuA and DeSiDeRaTa, from the University of Texas, Arlington. We also helped Teknowledge, lead of the Quite team, get the QuO/TAO integrated software running in the Quite testbed. We also helped the University of Oregon debug their QuO/ASSERT integration.

We attended and presented a paper at ISORC '99, where we discussed QuOIN integration with SRI and discussed plans to integrate their bandwidth management component into QuOIN v3.

June 1999

We hosted a QuOIN review for DARPA in which we demonstrated the QuOIN integration efforts, including the Bottleneck example using Darwin, Remos and RSVP, the real-time example using TAO, the dependability example using Proteus and Ensemble and the ATC example using OODTE.

At the same time of the QuOIN review, we held a planning meeting for a possible Advanced Tactical Demonstration (ATD) with MITRE, The Open Group (TOG), AFRL and NSWC. This ATD is a possible technology transition opportunity for QuOIN.

Our major accomplishment during June 1999 was release of QuO and QuOIN version 2.0. This release included a vastly improved QuO Toolkit, including new and improved QDL languages (such as the first instantiation of the Connector Setup Language, CSL) and code generators, a more robust QuO kernel, the QuO instrumentation package and the first instantiation of the QuO gateway. QuOIN version 2.0 includes the following integration software packages:

- Availability, with the Proteus dependability manager and Ensemble group communication mechanism integrated into the QuO framework
- Resource management, with integrated Columbia's RSVP ORB, CMU's Remos, and CMU's Darwin
- Real-time, with QuO controlling and simplifying the use of TAO's real-time event channels
- Caching, from Georgia Tech

We attended a meeting in Dahlgren, Virginia, with the QUITE team and the HiPer-D research and development team from NSWC. During this meeting, we planned QuOIN 2.0 and QUITE R2A integration and transition to NSWC HiPer-D. We followed up on this meeting with NSWC personnel to provide more information on QuOIN and answer some of their questions.

We attended a workshop on XML and presented our QuO/QuOIN work and an integration plan with XML standardization efforts.

July 1999

During July 1999, we released QuO and QuOIN version 2.0.1. This release fixed some bugs reported by users of version 2.0 and included minor functionality improvements.

We also met with Washington University, St. Louis, to plan the contents of QuOIN versions 2.1 and 3.0 and to schedule the efforts moving forward. We developed the plan for QuOIN v3 and elicited the participation of other Quorum researchers. We submitted this plan to DARPA for review and feedback.

We worked with members of The Open Group (TOG), QUITE team member, at our Cambridge facilities to get their QuO/Recon Simulator demonstration software running.

August 1999

During August 1999, we began implementing a general-purpose version of the QuO gateway, based upon CORBA's pluggable protocols, as implemented in TAO. We developed an initial example using a simple transport mechanism. This version of the gateway should support instantiations of different gateways with less effort than the current gateway.

We also began planning the release of QuOIN version 2.1 and initiating plans for the functionality in QuOIN version 3.

We visited Kane Kim at UC-Davis to plan his QuOIN version 3 participation.

We also started talking to Boeing about integrating QuO into the Boeing OFP software under the Weapon System Open Architecture (WSOA) project. This software was already using TAO for real time event scheduling.

September 1999
In September 1999, we executed a source code agreement with The Open Group (TOG). Under the terms of this agreement, we delivered portions of QuO and QuOIN source code for them to port to NT.

We also hosted a meeting for NSWC personnel from the Hiper-D program, University of Illinois, and TOG. We presented QuO, QuOIN, and the dependability work, and showed a number of QuOIN demonstrations. We also held discussions of how they might be able to transition QuOIN results into their environment and programs. We also released to them a copy of the QuO and QuOIN software.

Also during September, we worked on almost every aspect of QuOIN, in anticipation of an early October release of version 2.1. We made major strides in the development of the QuO gateway, as well as improvements in resource management.

Washington University released TAO version 1.0, the commercialized, Open Source version of TAO. This version of TAO is the stable version on which we will base QuOIN 2.1.

We also began working on a couple of conference papers, one for ISORC 2000 and one for Middleware 2000.

October 1999
Much of October 1999 was spent preparing for the Quorum PI meeting held at the end of October. This included developing ideas for working groups that were held at the PI meeting and coordinating the working group planning with the leaders of several of the groups.

We continued development of version 2.1 of QuOIN aiming for an October release of the software. As part of this we continued working on the QuO gateway shell, which was developed using TAO's pluggable protocol mechanism. WUStL developed improvements to TAO's Dynamic Skeleton Interface (DSI) and Dynamic Invocation Interface (DII) components to support the QuO gateway. We developed a version of the RSVP gateway using the new pluggable protocol gateway shell.

We also installed, tested, and documented a new version of the QuOIN dependability component, which included three replication schemes:
- Active replication with pass first
- Active replication with majority voting
- Passive replication

We integrated CMU's Darwin resource reservation manager into the QuO testbed, developed adaptive QuO example applications using it and experimented with the results under changing network conditions. Darwin provides an alternative to the RSVP resource reservation mechanism that had previously been integrated with QuO. We also integrated and experimented with CMU's Remos 1.2 status monitor. The integrated Darwin and the new version of Remos became components of the QuOIN 2.1 release.

We installed the new version of TAO, TAO version 1.0, the commercialized, Open Source version. We also began updating the QuO C++ examples and the real-time integrated event channel example to work with TAO version 1.0, which has a different event channel service.

We also created, installed and documented a stable version of the QuOIN security component, which integrated QuO and NAI's OO-DTE software. This security component became a new component of the QuOIN 2.1 release.

We submitted a paper on the QuO connector language and the special configuration needs of QuO applications to ISORC 2000.

We helped The Open Group get the QuO version 2.0.1 software installed and running in their testbed. We also hosted a meeting with Bill la Forge (TOG) to discuss the QUITE team's idea for approaching framework development.

We attended and presented at the Quorum PI meeting in San Francisco on October 25-29. We made formal presentations on the AQuA dependability parts of QuOIN, on the Bandwidth Management dimension of QuOIN, and presented an update on the QuOIN Distributed Object Integration efforts.

We also led and participated in a number of the working groups. QuOIN project members led and assisted with the following working groups:
- Adaptive Real Time Systems group (Doug Schmidt, WUStL)
- Disseminating Network Resource Information group (John Zinky, BBN, and Peter Steenkiste, CMU)
- Dependability and Failure Management (Bill Sanders, UIUC)
- Software Engineering and Tools (Stuart Faulk, UOregon, and Joe Loyall, BBN)

We also had informal discussions with a number of researchers and participants, including Bryan Doerr and the group from the Weapon Systems Open Architecture (WSOA) project at Boeing. Bryan and we discussed plans for using the QuO adaptive architecture as a critical piece of the WSOA architecture.

November 1999

We released QuO/QuOIN version 2.1 on November 12 and announced it to the Quorum community via e-mail. QuO/QuOIN 2.1 includes the following modules:
- The QuO toolkit module (developed under the Open Implementation Toolkit project) provides the software for the QuO framework, development environment and documentation. This module includes the QuO runtime kernel, the QuO Quality Description Languages (QDL), the QuO graphical user interface (GUI), libraries of system condition objects and instrumentation support. The QuO toolkit works with the Visibroker and TAO ORBs.
- The QuO gateway provides software for plugging in transport-layer protocols and mechanisms and controlling them.
- The DIRM module provides capabilities for monitoring and managing network bandwidth.
- The AQuA module provides capabilities for monitoring and managing dependability.

- The Real-Time module provides capabilities for interfacing and controlling the real-time features of the TAO ORB and services.
- The Security module provides access control.

QuO/QuOIN version 2.1 includes software from BBN, Washington University St. Louis, the University of Illinois, Columbia University, Cornell University, Carnegie-Mellon University and the University of Utah (Flick 2.1 provides an alternate IDL compiler for TAO 1.0). Important new capabilities in this release included the following:
- Improvements to the QuO toolkit software, including to the QDL languages, instrumentation and error reporting (developed under the Open Implementation Toolkit project)
- Integration with TAO v1.0, the commercially available version of TAO, providing a real-time performance property
- A QuO gateway for integrating different transport layer protocols, mechanisms, and controls, plus an example gateway instantiation for the RSVP bandwidth management protocol
- Simplified interfaces for AQuA, plus additional replication strategies, including passive replication and active replication with voting
- Support for survivability, using AQuA-based notification interfaces to recognize anomalous situations (developed under the Open Implementation Toolkit project)
- Support for reserved bandwidth through either RSVP or CMU's Darwin system
- Network status monitoring, through integration with CMU's Remos V1.2
- Integration with Network Associates' Object-Oriented Domain Type Enforcement (OO-DTE) access-control software, providing a QuO-controlled security component
- A demonstration of high-performance CORBA stub generation, through integration with Flick
- More examples, demonstrations and documentation

Also during November 1999, we attended a day long teleconference with the WSOA participants, including Boeing, WUStL and Honeywell. The purpose of the teleconference was to firm up the requirements for the WSOA system and to define the roles of the different participants and components, such as TAO (WUStL), RTARM (Honeywell) and QuO (BBN), in it. We followed up the teleconference with individual interactions with Bryan Doerr from Boeing, Chris Gill from WUStL and John Shackleton from Honeywell to sketch out the requirements, design and scenarios of the WSOA system and the benefit that QuO can add to it.

We began writing a paper, entitled "Flexible and Adaptive Control of Real-Time Distributed Object Middleware", for submission to *Real-Time Systems, The International Journal of Time-Critical Computing Systems, Special Issue on Flexible Scheduling of Real-Time Systems*. This paper described the integration of the QuO adaptive computing architecture and the TAO real-time ORB, and the way in which QuO can provide higher level control and adaptation of TAO real-time mechanisms.

December 1999
We concluded the paper, entitled "Flexible and Adaptive Control of Real-Time Distributed Object Middleware", and submitted it to *Real-Time Systems, The International Journal of Time-Critical Computing Systems, Special Issue on Flexible Scheduling of Real-Time Systems*. This paper describes the integration of the QuO adaptive computing architecture and the TAO real-time ORB, and the way in which QuO can provide higher level control and adaptation of TAO real-time mechanisms.

We began preparation for the DARPA Quorum demonstrations in Washington held in February 2000. We planned a demo consisting of an integrated system showcasing managed bandwidth and resource reservation, real-time, dependability and security all within the adaptive QuO framework. We began developing the demonstration using BBN's OpenMap™ software as a QuO application.

We performed some initial timing measurements of the RSVP instantiation of the QuO gateway. These were performed with no optimization and with debugging enabled. The gateway added about 30% performance overhead. In addition, we began design of a naming scheme to work with the QuO gateways. The naming scheme supports the existing CORBA references (IORs) on the client and server side, but enables finding and registering QuO objects, such as gateways, so that client-side references to remote objects get delivered transparently to gateways and routed to the proper server-side gateway.

We also participated in WSOA teleconferences and architecture reviews during the month of December. Boeing's Weapon System Open Architecture (WSOA) is a technology transition target for the Quorum program. QuO and TAO are two of the primary planned components for the WSOA architecture. We participated in requirements and design of WSOA to a) simplify Boeing's efforts to use and integrate QuO and TAO and b) to help extract WSOA requirements that focus QuO and QuOIN research and design.

January 2000

During January 2000, we continued development of BBN's planned demonstration at the DARPA Quorum Technology Demonstration in Washington on 7-11 February 2000. The demonstration was structured as a single client application, based upon BBN's OpenMap™ application (http://openmap.bbn.com), connected to a number of server objects using the QuO middleware. The OpenMap requested and received mapping data from the server objects, each of which exhibited specific QoS characteristics and control.

During January 2000, we achieved the following for the demonstration system:
- We designed the architecture of the demonstration system
- We identified the hardware and software components required
- We began developing the OpenMap client application
- We acquired the newest versions of the component software, including the TAO ORB, CMU's Darwin and Remos, UIllinois' Proteus and NAI's OO-DTE and began integrating them into the QuO framework within the context in which earlier versions have been integrated

We also began working on preparations for the other parts of the Quorum Technology Demonstration, including our QuOIN presentations and posters that will accompany our demonstration.

We continued to work with Boeing for transition of the QuOIN technology to the WSOA project. Boeing was in the requirements phase at that time and was working on a firm requirements document and preliminary schedule. We participated in these, identifying requirements for QuO/QuOIN components of the WSOA architecture and a preliminary schedule for providing and integrating them.

Finally, we also worked on firming up plans for the QuO/QuOIN version 3 components, research and integration.

February 2000

The primary activities in February 2000 centered around our preparation for and participation in the DARPA Quorum Technology Demonstration in Washington on 7-11 February 2000. During the course of these days, we presented several hour long and half-hour long demonstrations of the integrated QuOIN environment, as well as formal QuOIN presentations and posters illustrating the QuOIN architecture, the demonstration application and the Quorum components illustrated by the demonstration.

Our demonstration scenario illustrated a command and control application from the point of view of an AWACS aircraft mapping a battlefield. The AWACS aircraft queried remote tracking systems for position data and displayed the data on a map. The QuO framework software was used to integrate the client application and server objects and to manage QoS between the CORBA components of the system. For the demonstration, each tracking system exhibited a different QoS property, as described below.

The demo client application was based on OpenMap™, a geographic map display program (http://openmap.bbn.com). OpenMap makes CORBA calls to a layer-server and receives lists of glyph-objects to display on the map. Basically, the OpenMap client asked which glyph-objects the layer-server had within a particular rectangle (in lat/long coordinates). The layer-server returned the list of glyph-objects, which contained the coordinates and shapes (i.e., bitmaps). The OpenMap client integrated multiple layers into a single map, by drawing the layers over one another.

The demonstration architecture demonstrated heterogeneity in operating systems (Solaris and Linux), languages (Java and C++), ORBs (Visibroker and TAO) and network resources (1 Mbps, 10 Mbps and 100 Mbps links). The application demonstrated the fusing of data from remote sensors across a wide-area network (WAN) into a single, integrated map. The OpenMap client continuously requested data from the layer-servers (representing remote sensors) and redrew the map.

The terrain map layer demonstrated *managed bandwidth*. The terrain data server lay several hops from the client, separated by several routers and different bandwidth network segments. During the demonstration we flooded one of the 10 MB links along the path with cross-traffic data, reducing the bandwidth available to respond to the terrain requests. Our demonstration system detected the bottleneck (using CMU's Remos system) and illustrated two types of responses. First, the application responded by requesting lower bandwidth terrain data (with lower resolution). Also, when it was able, the application reserved bandwidth to transport the terrain data using CMU's Darwin system.

A layer displaying ground troops illustrated *dependability*. The ground troops were protecting a bridge and were tracked by replicated sensors on five different remote machines. To demonstrate the dependability and fault tolerance of the system, we killed some of the sensor processes and injected erroneous data into some of the sensors. In every case, the mapping client continued to update the ground troop data accurately and the system recovered by restarting killed sensors and killing and restarting malfunctioning sensor processes.

A layer displaying air traffic illustrated *security*. Simulated air traffic control sensors tracking aircraft positions were protected behind an OO-DTE access control firewall. Intrusion detection systems (IDSs) reported unauthorized data access (e.g., manipulation of sensor data) and sensor penetration (e.g., corruption of the sensors themselves). When the IDSs triggered alerts, QuO triggered changes in OO-DTE access control policy to protect against unauthorized access.

A set of F-15s illustrated *real-time* QoS. We used the TAO real-time event channel to deliver position events tracking the location of two F-15s. We flooded the event channel with other events and used QuO to change the priority of one of the F-15's events to higher priority so that they pre-empted the lower-priority events and the position continued to be calculated and updated accurately.

In addition to the structured demonstration which we showed to Government and industry personnel many times over the 2½ days of the technology demonstration, we displayed several posters illustrating our research and integration, gave a formal presentation and participated in many informal discussions with attendees and other researchers.

Included in these were discussions with personnel from NSWC, with whom we are targeting inclusion of QuOIN components in their 2000 demonstration system. We talked to Mike Masters, Leslie Madden and Paul Werme about how we can support their efforts.

In addition to the Quorum Technology Demonstration, we also continued working with Boeing for transition of the QuOIN technology to the WSOA project. We agreed on a set of requirements for QuO's inclusion in the WSOA demonstration system and a schedule for providing it. We have been working very closely with Boeing for months to help develop the requirements for their system, the schedule and to identify the roles for QuOIN technology in it.

The weeks leading up to the Quorum Technology Demonstration were filled with efforts ranging from development of presentation and poster material, technical integration of software from the University of Illinois, CMU, Cornell, Washington University and NAI (during which we worked tirelessly with the teams from these organizations) and scenario and demonstration development.

March 2000

The primary activities during March 2000 centered on beginning development efforts in support of transition to the Weapon Systems Open Architecture (WSOA) project, writing a submission to a journal and planning for QuOIN v.3.

Transition to the WSOA project. Boeing is transitioning three pieces of Quorum software into their WSOA ground demonstration in January 2001 and air demonstration in January 2002. They are the QuO adaptive framework from BBN, the TAO real-time ORB from WUStL and RT-ARM from Honeywell. Figure 6.1 illustrates how QuO contracts (the parallelogram), QuO system condition objects (the pentagons), QuO delegates (the sideways pentagon), the TAO ORB and the RT-ARM Resource Manager fit into the WSOA architecture.
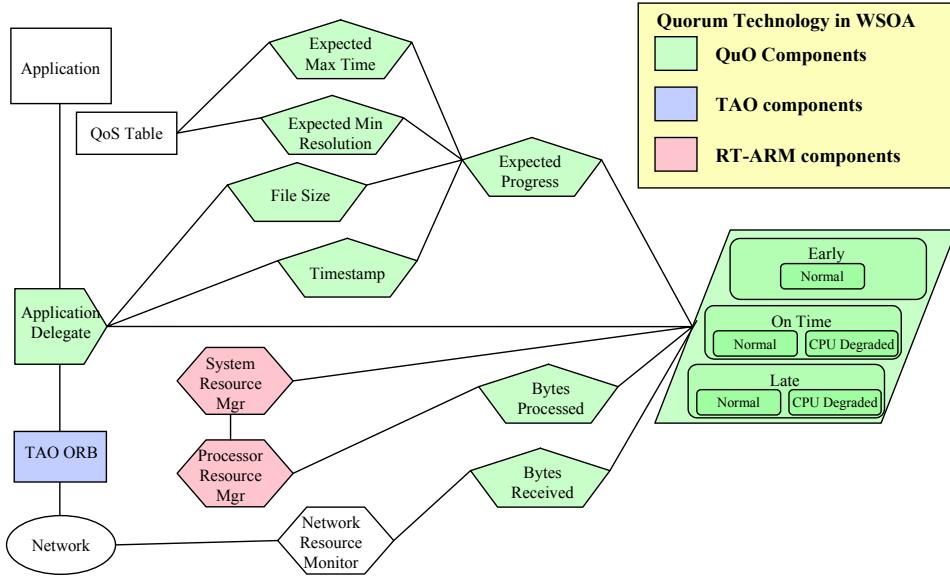
Figure 6.1: WSOA Architecture

In support of these activities, BBN, WUStL and Boeing developed a set of requirements needed from the QuO software. Many of these requirements, described in the following tables, already exist in the current implementation of QuO (indicated by a *No* in the *New/modified for WSOA* column). Others require modification to, or enhancement of, the existing QuO software (indicated by a *Yes* in the *New/modified for WSOA* column).

General QuO Framework Requirements

Requirement	New/modified for WSOA
The QuO framework shall compile and execute in C++.	Yes
The QuO framework shall be functionally optimized.	Yes
The QuO framework shall use ACE to encapsulate all OS services.	Yes

QuO Delegate Requirements

Requirement	New/modified for WSOA
The delegate shall provide the same interfaces as the remote object it represents.	No
The delegate shall trigger contract evaluation upon each method call on the remote object (pre-method evaluation).	No
The delegate shall use the current operating region returned from contract evaluation to choose how to continue processing the method call on the remote object.	No
The delegate shall be able to perform adaptive behavior when processing a method call on the remote object.	No

The delegate shall be capable of providing an interface for periodic dispatch.	No

QuO Contract Requirements

Requirement	New/modified for WSOA
The contract shall provide the current operating region.	No
The contract shall provide transition behavior to invoke upon switching operating regions.	No
The contract shall define the operating regions possible and their boundaries.	No

QuO System Condition Object Requirements

Requirement	New/modified for WSOA
A system condition object shall provide storage of a current value.	No
A system condition object shall provide an interface for retrieval of its value.	No
A system condition object shall provide an interface for updating of its value.	No
A system condition object shall provide an interface for initializing its value.	No
A system condition object shall be either observed or non-observed.	No
An observed system condition object shall notify the passive QuO kernel upon a change in its value.	Yes

The original version of QuO was developed with the following goals: rapid prototyping, maximum flexibility and maximum functionality. The resulting implementation was written in Java, distributed and multi-threaded. As indicated in the above tables of requirements, WSOA requires: complete control over threading, a small footprint and low overhead and a complete C++ implementation.

During March 2000, we began designing a passive QuO Kernel to fulfill these requirements. This kernel does not contain any separate threads, i.e., it runs in the thread of the caller, which should be either a delegate evaluating a contract during a method call (*synchronous* or *in-band*) or an *observed* system condition object whose value has changed (*asynchronous* or *out-of-band*).

We also wrote and submitted a paper entitled "Coordinated Middleware-Based End-to-End Qos Management for Next-Generation Distributed Applications" and submitted it for publication in the Computer Communications special issue on QoS-Sensitive Network Applications and Systems. This paper describes some of the issues involved in our work in the areas of end-to-end adaptive real-time behavior using TAO and QuO and presents some of our results to date.

We also began our planning for the QuOIN v.3 release. We plan to improve the usability of the QuO toolkit software, enhanced bandwidth, resource management capabilities, dependability and real-time capabilities, and combinations of security and dependability, among other capabilities.

April 2000

During April 2000, we continued development efforts in support of transition to the WSOA project. In April, Boeing identified another Quorum component that they planned to transition into WSOA, identifying Oregon Graduate Institute's (OGI) Quasar project to provide network resource management on the C2 side of the demonstration.

During the month of April, we began developing the passive, completely C++ version of the QuO infrastructure, i.e., the kernel, contracts and system condition objects that we designed in March.

Also during the month of April, we prepared for the May 2000 Quorum PI meeting in Seattle, WA. We made plans to present, along with Boeing, Honeywell, WUStL, UCI and OGI, information about the WSOA technology transition effort. During April, we began coordinating and preparing this presentation.

May 2000

During May 2000, we continued development efforts in support of transition to the Weapon Systems Open Architecture (WSOA) project. We participated in meetings to focus technology transition efforts to the NSWC HiPer-D program. Finally, we attended and participated in the Quorum PI meeting in Seattle, WA, May 8-11, 2000.

WSOA Transition. During May, we continued the development of the passive C++ version of QuO. This version of QuO fulfills the requirements that WSOA has for the QuO adaptive engine: smaller footprint, higher performance, C++ implementation and no threading. We completed initial development and began testing this version of QuO in anticipation of delivery to Boeing early June 2000.

NSWC Transition. We hosted a visit by Gary Koob, DARPA Quorum program manager, and Todd Carr to BBN during May 2000 to discuss technology transition efforts to the HiPer-D program (NSWC). This meeting was followed with a brainstorming meeting at STDC in Herndon VA, which we attended and participated in, and which was also attended by representatives of DARPA, NSWC, and the QUITE integration team (Teknowledge, OpenGroup, and STDC). This meeting identified a group of candidate technology transition efforts, some of which were already underway.

We are primarily involved in two of the technology transition efforts: AQuA for fault tolerance and the UAV scenario. The AQuA transition is ongoing and involves Leslie Madden and Tracy McDonald from NSWC, BBN, and the University of Illinois. One major focus currently is AQuA support for multi-threaded applications.

The UAV scenario is a streaming video application in which a video feed from an off-board unmanned autonomous vehicle (UAV) is distributed to multiple on-board video displays and QoS of the video images is maintained using QuO and resource management based on DeSiDeRaTa. We talked to Paul

Werme (NSWC) during the meeting in Herndon about the basic concepts, followed it up with a preliminary architecture document that we sent to Paul after the Herndon meeting and then held detailed architecture discussions with Paul and others at the Quorum PI meeting.

Also during the month of May, we finished our coordinated presentation for the Quorum PI meeting with Boeing, Honeywell, WUStL, UCIrvine and OGI. We presented this talk, describing the WSOA technology transition effort, during the Quorum PI meeting on May 9, 2000 in Seattle WA.

Also at the Quorum PI meeting, we participated in a number of working groups including leading the Portability and co-leading the Failure Notification working groups. We also participated in the Layered Resource Management working group, where we helped develop the UAV scenario, which will integrate QuO and DeSiDeRaTa within the HiPer-D testbed. Finally, we also attended the QoS Open Testbed working group.

June 2000

During June 2000, we continued development efforts in support of transition to the Weapon Systems Open Architecture (WSOA) project. We commenced development efforts in support of transition to the NSWC HiPer-D program. Finally, we continued planning and development on QuOIN version 3.

WSOA Transition. During June 2000, we delivered an initial version of the QuO/C++ (passive) implementation, with an example, to Boeing. In the process of testing this code in the Boeing environment, we discovered an issue with differences in the versions of ACE and TAO being used in the WSOA effort and those being used by BBN in the development of QuO and QuOIN. BBN used the latest (ACE 5.1.4 and TAO 1.1.4) versions of ACE and TAO to develop the QuO kernel, while Boeing is using a modified version of ACE 5.0.7 and TAO 1.0.7. This resulted in the CORBA stubs used by the Boeing code to access QuO kernel features (generated by TAO 1.0.7) being incompatible with the stubs used internally by the QuO kernel (generated by TAO 1.1.4). We explored options for overcoming this with Boeing and settled on a process in which they provided BBN with their environment (i.e., the modified ACE 5.0.7 and TAO 1.0.7 libraries and code). BBN continued core development using the latest code, but builds Boeing special-purpose releases in their environment. The first time we did this, however, we encountered run-time errors using the Boeing environment that do not occur in the BBN environment. These errors turned out to be configuration problems and known errors that we corrected and worked around in close cooperation with Boeing.

NSWC Transition. Also during June 2000, we began efforts to develop the UAV scenario, code and demonstration for the NSWCDD HiPer-D testbed. We followed up our discussions with NSWC at the Quorum PI meeting with e-mail and voice exchanges to iterate over an architecture and scenario for the UAV portion of the demonstration. At the same time, we began identifying and acquiring candidate video applications that could serve as a basis. The AV streaming service of the TAO code includes an example application based upon the OGI Quasar MPEG player. The TAO AV streams code adds a CORBA control interface to the Quasar application. However, the TAO AV streams example had not been maintained, so it was no longer working. BBN and WUStL worked together to get the application up to date and to fix the bugs in it. By the end of June, we had the MPEG player working and had portions of QuO integrated into it.

We also provided a schedule to NSWCDD, including a planned visit to deliver, install and demonstrate an initial capability in mid-July, followed by discussions planning the rest of the scenario and demonstration development.

Also during June 2000, we continued work on bandwidth management, gateway shell and infrastructure improvements for QuOIN version 3. We visited UIUC to work on common development of the common gateway shell based on ACE/TAO and the specific handlers needed for AQuA dependability. We also began implementing the ACE/TAO version of the AQuA gateway for Maestro.

July 2000

During July, we continued development efforts in support of transition to the Weapon Systems Open Architecture (WSOA) project and to the NSWC HiPer-D program and continued development on QuOIN version 3.

WSOA Transition. During July, we discovered the source of the runtime anomalies that we were encountering in our QuO example running in Boeing's environment. These were due to two problems, which we were able to fix and to work around. First, server objects in the Boeing-tailored version of TAO needed to have the –ORBEndpoint command line argument explicitly set. Second, the Boeing-tailored version of TAO did not exit from the orb->run() loop gracefully. In each case, Boeing helped identify the known problem and indicate to us a way to work around it.

Upon overcoming this hurdle, we were able to deliver a new version of the QuO runtime implementation and example that Boeing was able to install and run out of the box.

NSWC Transition. During the month of July, we continued development of software in support of the NSWCDD HiPer-D UAV demonstration. We developed an improved version of a streams-based video distribution and player with a CORBA control layer, and a QuO control engine that synched up two video displays when one falls behind the other (e.g., if one video display machine becomes overloaded). We visited NSWC, installed and demonstrated this software. We also held discussions to focus the following month of effort towards developing the proper software for their demonstration. These discussions and interactions were fruitful and we entered the last part of July with a plan to proceed for the month of August. This plan was detailed by e-mails exchanged between BBN and NSWC and copied to DARPA.

QuOIN version 3 Development. We made major strides toward QuOIN version 3, and toward QuOIN goals during July. First, we developed the first full implementation of the QuO code generators that runs on Windows NT. This, along with the Windows NT version of the QuO C++ kernel reported last month (and above), means that we now have a full version of QuO for the Windows NT platform. We delivered the code generators to Boeing, who installed and ran them successfully.

We also continued working on solutions for a new AQuA gateway, based upon the QuO gateway shell, which in turn is based upon TAO's pluggable protocols. The old AQuA gateway includes handler code that is tightly intertwined with the old gateway code. We are working on a solution that abstracts the handler code so that it can be plugged into the new gateway. We also built a performance-test harness for the QuO gateway, and collected initial performance numbers, which indicated that the gateway imposes less than a one millisecond overhead. We also planned another

visit to the University of Illinois to work closely together with them on this AQuA gateway implementation.

Two new releases of CMU Remos (1.3, 1.4) were integrated into the QuOIN Bandwidth management. We conducted extensive testing of both the Remos SNMP and WAN pollers and incorporated bug fixes into Remos Release 1.4. The goal of the Remos testing is to make a system that is capable of monitoring a small Enterprise-size network, i.e., 20 hosts at 5 sites separated by a Wide Area Network.

Finally, we designed some major improvements to the QuO QDL languages, targeting their implementation for QuO/QuOIN version 3.

August 2000
During August, we continued development efforts in support of transition to the Weapon Systems Open Architecture (WSOA) project and to the NSWC HiPer-D program and continued development on QuOIN version 3.

WSOA Transition. During August, we attended the WSOA quarterly review meeting at Boeing in St. Louis, Missouri. As part of this, we prepared and presented the part of the quarterly review briefing dealing with the QuO adaptation middleware and QoS control. We also stayed for a few days at Boeing to help develop the QuO contract, delegate, and connector code for WSOA. Honeywell and WUStL were also at these meetings and the working sessions, working on the RT-ARM resource manager and TAO adaptive scheduler, respectively. This gave us the opportunity to discuss and design some of the interfaces between QuO and these parts and to build the software (i.e., system condition objects) implementing these interfaces.

NSWC Transition. During the month of August, we proceeded implementing the pieces of software for the NSWC UAV demonstration software that we had agreed upon during and immediately following the visit to NSWC during July. As a result of these meetings with NSWC, we started working on the following tasks:
- Install 3 stage pipeline (video source, video distribution, video displays)
    - -Get video displays to read MPEG from a stream instead of files
    - -Get video distribution process to send out two streams
    - -Get video distribution process to get its input from a stream
    - -Get video distribution process to drop frames along one stream
- Provide 4 Linux PCs in test bed
- Develop QuO contracts, system condition objects, and delegates
    - -For video displays
    - -For video distribution
- Support for NSWC instrumentation and RM adaptation
    - -Make sure that video distribution process can be killed gracefully
    - -Make sure that video distribution process can be restarted
- Develop load mechanism/procedures for video distribution and displays
- Visit NSWC to install and demo software

QuOIN version 3 Development. During August, we continued development of a new AQuA gateway, based upon the QuO gateway shell, which in turn is based upon TAO's pluggable protocols. The old AQuA gateway includes handler code that is tightly intertwined with the old gateway code. We are working on a solution that abstracts the handler code so that it can be plugged into the new gateway. We visited the University of Illinois for another joint development meeting, during which we finalized a number of design issues, demonstrated a TAO+Maestro application with improved performance and coded most of a gateway shell capable of running the CORBA-forwarding handler and loading it dynamically.

September 2000
During September, we continued development efforts in support of transition to the Weapon Systems Open Architecture (WSOA) project and to the NSWC HiPer-D program and continued development on QuOIN version 3.

WSOA Transition. During September, we made further enhancements to the QuO infrastructure to support the WSOA technology transition. We protected the QuO contract execution in the passive C++ kernel with ACE locks. This enables the QuO developer to choose to use contracts that are shared by different execution threads while avoiding race conditions. We also developed a stubbed-out version of the WSOA application at BBN so that we could test the QuO functionality in a WSOA context, without the Boeing- and Honeywell-specific components. This was to ease the debugging burden on Boeing upon integration of QuO into the WSOA environment. We also enhanced the QDL code generator to generate additional code required for the WSOA application.

NSWC Transition. We visited NSWC during August 22-24 to install and transition a new version of the UAV demonstration software. This new version addressed all of the issues in the development schedule discussed in last month's report. It included a 3-stage pipeline (video source, video distribution, video displays), which sends MPEG video as a stream. It included QuO contracts and system condition objects that recognized excessive load on the hosts in the second and third stages of the pipeline (i.e., the distributor and the video displays) and adapted the distributor and displays to drop frames to compensate. The software also had the capability to gracefully kill the distributor process and restart it on another host under conditions of unacceptably excessive load.

During installation and demonstration at NSWC, NSWC provided another, more extensive, list of enhancements and requirements for the UAV demonstration software.
The major items that we were addressing during this iteration on the software involved the following:
* Obtaining and providing direct measurements of the distributor performance to the NSWC instrumentation package (i.e., actual frame rate, expected frame rate, input queue size, output queue size and number of dropped frames each second).
* Basing the QuO contract on the distributor host (contracts on video display hosts was no longer desired) on internal measurements, such as the queue length and/or actual frame rate.
* Synchronization of the startup of a new distributor and stopping of the old distributor.
* A number of smaller enhancements, tweaking, and minor fixes based upon testing and demonstrating in the NSWC HiPer-D testbed.
We provided a new version of the software to NSWC on September 15 that addressed the list of items that NSWC requested. It measured queue lengths and frame rates and passed them to the NSWC

instrumentation. It had a contract that adapted to drop frames when the queue lengths increased and used CPU load improving as a trigger to discontinue frame dropping. A new distributor could be started and the old one stopped anytime after starting the new one. We also address the other minor issues that NSWC requested. We followed up the electronic delivery of this software with an on-site visit to NSWC to aid in integration and testing on September 21-22.

QuOIN version 3 Development. During September, we continued development of the new AQuA gateway and on making the dependability manager dependable. This included development of the top part of the AQuA gateway (over the handlers). Toward the end of the reporting period, messages were able to flow through the new architecture of the gateway. Then we began testing the new implementation. At the end of the calendar month of August, we began working on the re-implementation of the passive replication handlers in the new gateway. We first worked on the message flow when no failure occurs. Simultaneously to this, we worked on making the dependability manager dependable. This is done by slightly changing the group structure and adding new communication features. In order to verify the correctness of our ideas, we implemented the design in the former gateway architecture.

We also made numerous improvements in the QuO infrastructure, including the QuO QDL languages and runtime support. Many of these were motivated by needs of the UAV or WSOA technology transition efforts. We also developed capabilities for a resource status service, for measuring and analyzing system information, which will become an important part of QuO/QuOIN v.3.

Finally, we authored a submission, entitled "Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications", for the 21st IEEE International Conference on Distributed Computing Systems (ICDCS-21). This paper, which we co-authored with Chris Gill of Washington University, St. Louis and Jeanna Gossett of Boeing, describes the technology transition efforts to the WSOA and NSWC environments. It compares and contrasts the different use-cases and environments in which QuO can be used and the support that we've developed in QuO for those environments.

October 2000

During October, we continued development efforts in support of transition to the Weapon Systems Open Architecture (WSOA) project and to the NSWC HiPer-D program and continued development on QuOIN version 3.

On 25 and 26 September, we hosted a visit by Thomas Lawrence and Patrick Hurley of AFRL. We presented a review of QuOIN and other project activities, including the progress we've been making in the technology transition activities to the WSOA effort and the NSWC UAV demonstration, the resource status service and the dependability and gateway activities.

We also hosted a visit by Doug Schmidt (DARPA/ITO) on 27 September, in which we demonstrated the UAV software.

WSOA Transition. During October, we helped Boeing with the integration of QuO code into the F15 application. Boeing reported success in the integration process and began integrating additional pieces

and testing. We made a few fixes and enhancements to the QuO code to better support the WSOA example. Finally, we began the process for providing Boeing access to the QuO source code to aid in their debugging and integration efforts.

NSWC Transition. We visited NSWC again on 21-22 September to aid in integration, demonstration and testing the UAV software in the NSWC HiPer-D testbed. In the course of testing and demonstrating the software in the HiPer-D testbed, we identified a list of issues that needed to be addressed, which fell into three basic categories:

- Tweaking, tuning, and debugging for specific aspects of the NSWC environment (e.g., to interface to their process manager, graphical package, etc.)
- Measurements of the internal distributor behavior as reported to the instrumentation. Testing in the HiPer-D testbed with the NSWC graphical interface illustrated that the distributor queues were frequently empty and that the actual frame rates and number of dropped frames varied within a wide range.
- Tuning of the contract region definitions. NSWC required that the contract regions be defined in terms of internal distributor measurements only, i.e., queue lengths or actual frame rate, rather than external measurements, such as CPU load. However, once a transition occurred because of something such as degraded actual frame rate, it's necessary to have a trigger back to normal behavior. The actual frame rate will never improve when frames are being dropped, even if the source (e.g., CPU load) of the degradation disappears. Furthermore, the trigger for transitioning needs to eliminate frequent transitioning between regions (thrashing). We discuss below the solution that we developed for solving this problem and meeting NSWC's needs.

We addressed all of the issues in the first and second categories. The tuning of the contract definitions required designing a method for transitioning when conditions degrade, then back when they improve, while avoiding thrashing. We began designing and implementing improved contract definitions. We discussed several possibilities with NSWC and began implementing some of them.

QuOIN version 3 Development. During October, we continued development of the AQuA gateway. We tested the top part of the gateway and fixed several bugs. One bug required significant effort and is still not fixed. Even with this remaining bug, we decided to continue the implementation. We implemented the active replication pass first handler during the report period. We will test the new handler next report period. We also continued our work on the passive handler in the new gateway. After the implementation of the architecture for the flow of messages, we worked on the buffers in order to tolerate crash failures. We decided to test the implementation at various steps.

November 2000

During November, we continued development efforts in support of transition to the Weapon Systems Open Architecture (WSOA) project and to the NSWC HiPer-D program and continued development on QuOIN version 3.

WSOA Transition. During November, we continued working with Boeing on the Build 1 version of the WSOA software. We provided Boeing with the QuO source code, along with the support that they needed to build it under Windows NT. During November, Boeing successfully tested the application functionality in which an F-15 collaboration client *getImage()* call is converted by the delegate into *getTile()* calls.

NSWC Transition. We visited NSWC again from 17-19 October to deliver, integrate, and demonstrate the fixes that we made to the software to address the list of issues identified in the previous month. Over the three days, we worked with NSWC to install, tune and test the software. At the end, NSWC was satisfied that most of the issues had been resolved, with the exception of two remaining issues to be addressed:

- *Hysteresis* - The application can get into a state in which region transitions occur frequently and rapidly
- *Jitter* – The video player exhibits jitter, displaying frames in a burst then pausing, when some adaptation is performed

We suggested several options to NSWC for controlling hysteresis. NSWC requested a solution in which we make sure that the contract stays in a region for a minimum amount of time before being allowed to transition to a more degraded region. At the same time, the contract, after being in a region for a specific amount of time, would try to improve things (i.e., it would try to send frames through at a higher rate) and would transition to a higher region if the experiment succeeds. In order to implement these, we began implementing timer system condition objects and contract subregions that would trigger the *testing* behavior (i.e., trying to send more frames through).

We added a few features to QuO's contract description language, CDL, to support the contract structure that NSWC needed for integration with HiPer-D. First of these is an **until** clause in regions (and states, described next). The **until** clause places a predicate on when a contract can leave a region once it's entered it. The second feature is support for **state** declarations. States are alternatives to regions that implement true state machines, with predicates on transitions. Using these, we implemented and tested a QuO UAV contract that degraded smoothly, dropping frames along the way and then periodically tested whether conditions have improved, in order to move to a better region.

For the jitter problem, we began conducting experiments to try to determine the source of the problem. We quickly determined that the source of the jitter lay within the video player itself. That is, we instrumented the video pipeline and ran a number of experiments in order to determine under what conditions the problem occurred and at what stage in the pipeline things went awry. The first two stages in the pipeline behaved as expected, but under certain conditions, the third stage, i.e., the video player appeared to get confused and enter a state in which it behaves erratically, buffering frames, then dumping a bunch of frames all at once, then pausing (while it buffered more frames).

These experiments helped us track down the source of the jitter of the video player upon restart, and fix it. Basically, the video player, which is off-the-shelf software written originally to play MPEG files, had code in it (deep in the display code) to compensate for slow video cards (no longer an issue). Starting a new distributor confused it (it believed that it had fallen behind in displaying the MPEG file) and it entered a mode of extreme jitter. We were able to fix this and scheduled to deliver the new software to NSWC.

QuOIN version 3 Development. During November, we continued development of the AQuA gateway. We continued to try to track down the bug reported in last month's report. We began testing the new active replication pass first handler and continued our work on the passive handler in the new gateway.

We also continued work on the QuO languages, runtime support and runtime services. Key among these was a new organization of the QuO languages, based upon the usability lessons we've learned, and a new version of the SDL, which is more aspect-oriented.

We attended the first meeting of the Quality of Service Task Force, sponsored by the Open Group, on 25-26 October.

December 2000

During December, we continued development efforts in support of transition to the NSWC HiPer-D program and to the Weapon Systems Open Architecture (WSOA) project and continued development on QuOIN version 3.

NSWC Transition. We provided fixes for the two last remaining issues NSWC had with the UAV software, i.e., hysteresis of contract region transitions and jitter of the video display upon distributor restart. We delivered the new software to NSWC electronically on 16 November. NSWC successfully installed the software and demonstrated it internally on 29 November. The UAV software was demonstrated as part of the external HiPer-D demonstration for the first time on 5 December.

In addition, the University of Illinois began discussions with the HiperD Team on how to incorporate the AQuA framework into their 2001 demonstration.

WSOA Transition. On 8 December, we attended and participated in a WSOA quarterly review at Boeing in St. Louis. During this meeting, we presented about the QuOIN technology transition efforts in WSOA and participated in a demonstration of the WSOA Build 1 software.

Quorum PI meeting. We attended and participated in the Quorum PI meeting on 12-14 December in Orlando, FL. We gave a presentation on the QuOIN distributed object integration effort. We led the Middleware Working Group and participated in the other (Layered Resource Management, QoS testbed, and Multidimensional QoS management) working groups. In the Layered Resource Management working group, we led a working group breakout session on the UAV demonstration planning. In the System Management working group, we gave a presentation on our resource status service and composing QoS facets.

QuOIN version 3 Development. During December, we continued development of the AQuA gateway. We continued to try to track down the bug reported in last month's report. We began testing the new active replication pass first handler and continued our work on the passive handler in the new gateway.

We also continued work on the QuO languages, runtime support and runtime services. These included continuing work on the new organization of the QuO languages, based upon the usability lessons we've learned, and a new version of the SDL, which is more aspect-oriented.

Also during December, the University of Illinois continued working on the new AQuA gateway with the help of BBN. We continued testing the top part of the gateway (over the handlers), found several new bugs and fixed the bug reported one month ago. Another part of this period was dedicated to testing the active pass first handler. Preliminary performance results of the new AQuA gateway using

the active pass first handler show that the performance of the whole AQuA architecture will significantly be improved compared to the former implementation. We also continued working on the passive handler focusing on the features required for recovering from a crash failure.

January 2001

During January, we continued development efforts in support of a QuOIN version 3 release this spring. We also continued our support of transition to the NSWC HiPer-D program and to the Weapon Systems Open Architecture (WSOA) project.

QuOIN version 3 Development. During January, we continued work on the QuO languages, runtime support and runtime services. These included continuing work on the new organization of the QuO languages, based upon the usability lessons we've learned, and a new version of the SDL, which is more aspect-oriented. We completed an initial implementation of the base aspect-oriented version of SDL supporting Java. We began moving our examples and demonstrations to the new language organization and SDL.

Also during January, we continued development of the new AQuA gateway. We finished the implementation of the passive handler (the state is multicast in the replication group after each output message). We started an extensive testing phase of the current version of the AQuA system (new gateway with the active pass first and the passive handlers) using several applications: a simple pinger application, the "deet" application developed by P. Rubel, and the castle application developed at UI and demonstrated in several DARPA meetings. Preliminary performance results indicate that the new version of AQuA gives better results than the previous version. We also continued working on replicating the dependability manager.

NSWC Transition. We continued discussions with NSWCDD about how to incorporate the AQuA framework into their 2001 HiPer-D demonstration.

WSOA Transition. During January, Boeing sent us the QuO code they are using in their current build of the WSOA software. We plan to look at updating it to the newest version of QuO next month.

February 2001

During February, we continued development efforts in support of a QuOIN version 3 release this spring.

Our main work in February consisted of finalizing development work on QuOIN version 3 and updating all example applications to use the new language and runtime versions. We finished up work on the new language organization, which enables QoS aspects to be separately implemented as reusable elements, called *Qoskets*. We also unified the three versions of the QuO runtime kernel: C++, Java CORBA and Java RMI, offering the configuration choices illustrated in Table 1.

	Java CORBA	Java RMI	C++ CORBA
Integrated	Yes	Yes	Yes
Non-integrated	Yes	Yes	Yes
Threaded	Yes	Yes	No, 3.1
Non-threaded	Yes	Yes	Yes

Table 1: Configuration choices for the QuO 3.0 kernel

As indicated in this table, we provide mix-and-match versions of the QuO kernel configurations, with the exception of the threaded version (i.e., the kernel has its own implementation thread) for C++. We plan to provide this in a future version of the software. The current version of the software includes examples that illustrate how to achieve the effect of a threaded QuO kernel, using an observed system condition object to provide the threading control.

We also finished up work on the Resource Status (RSS) and StatusTEC (typed event channel) services that are to be released with QuOIN version 3.0. We developed these two services, RSS and StatusTEC, to aid in the collection and dissemination of resource status information throughout a distributed application and the wide-area network of resources on which it can be hosted.

Also during February, we continued working on the new AQuA gateway. We continued working on an extensive testing phase of the new version of the AQuA system (new gateway with the active pass first and the passive handlers) using several applications: a simple pinger application, the "deet" application. We plan to continue testing the system using the castle application. We also began working towards a release (e.g., make files, scripts).

We also finished working on replicating the dependability manager. We have then proceeded with a testing phase. Finally, we have started analyzing the performance of the new AQuA system with the replicated dependability manager. The preliminary results show that the performance of the new AQuA system is better than the former version. Active replication also leads to a higher performance than passive when the state is multicast after each output message. Finally, we saw that the recovery process from a dependability manager failure has a performance equivalent to any replica failure recovery.

Technology Transition. During February, we continued to work with Boeing to support their upcoming quarterly review and associated demonstration. We began building a version of the WSOA software (with many of the Boeing specific pieces stubbed out) at BBN to ease transitioning the software to QuOIN version 3.0. However, we postponed these activities since Boeing indicated that they didn't want to move to version 3.0 until after the March quarterly review. Instead, we concentrated on helping Boeing make sure that the March demonstration was fully supported by the QuO software that Boeing already had.

March 2001
During March, we finished up development efforts in support of a QuOIN version 3 release this spring.

Our main work in March consisted of finalizing development work on QuOIN version 3, updating all example applications to use the new language and runtime versions, testing and packaging the release. We converted most of the QuO and QuOIN example code to the new language organization and tested it extensively. We incorporated nightly builds and regression tests of the software to ease the testing burden. We also produced new versions of the QuO/QuOIN users manual and reference manual for the v.3.0 release.

We began investigating the process to release the code as Open Source.

QuO/QuOIN v.3.0 has the following new features:
- Qoskets: mechanism for bundling QoS adaptive code for REUSE
- C++ integrated QuO Kernel
- Java RMI support
- Aspect SDL:
    - Language independent (C++ and Java) with Java-like syntax
    - Multiple SDL files can be woven together
    - SDL Templates to indirect Method signatures
- State Machine Representation for Regions in CDL
- Resource Status Service: Unified architecture for monitoring the status of external resources from inside an application
    - Default resource configurations published on web pages
    - QuO Status Typed Event Channel push technology for getting host load and capacity
    - Direct interface to Remos, for network monitoring
- Reusable in-band instrumentation, i.e., an instrumentation Qosket allows instrumentation independent of the Business interface
- Example Adaptive Code for:
    - Resource Monitoring with Client-side adaption (RSS and in-band instr)
    - Bandwidth Reservation (RSVP)
    - Security (Authentication)
- Easy to install Linux RPMs

During March, we continued work on AQuA, the one remaining development effort prior to release of QuOIN version 3.0. We continued working on the new AQuA gateway targeting inclusion with the QuOIN release. We continued working on an extensive testing phase of the new version of the AQuA system (new gateway with the active pass first and the passive handlers) using several applications: a simple pinger application, the "deet" application, and the castle application. We also started building the release structure (e.g., make files, scripts). At the end of the report period, we began installing the newest versions of AQuA software at BBN for testing and porting of QuOIN applications to the new version of AQuA.

April 2001
Our main work in April involved documentation, testing and packaging the QuOIN version 3 software. We instituted nightly builds and produced preliminary packages of the QuOIN software, which we handed off to several independent people, including people at BBN and at Washington State University, to test. We produced a User's Manual, a Reference Manual, installation and build

instructions and README files for all the example code. By the end of calendar month April, we considered the non-AQuA parts of QuOIN version 3 completely packaged and ready for release. We postponed actual release until May so that we could try to finish up the development on AQuA and produce the license and infrastructure for the open-source release of QuOIN.

During April, we also continued working on the new AQuA gateway. We continued working on an extensive testing phase of the new version of the AQuA system (new gateway with the active pass first and the passive handlers) using several applications: a simple pinger application, the "deet" application, and the castle application. The testing phase primarily focused on the implementation on Solaris. The University of Illinois sent a stable binary version to BBN for testing.

Finally, in April, we worked with Boeing to plan the remaining tasks requiring BBN participation under the Quorum program in support of the WSOA project. BBN has been transitioning QuO technology, developed under Quorum and QuOIN to Boeing. QuO is being used to help manage adaptation in dynamic mission planning in avionics systems. QuO, the TAO dynamic real-time scheduler, and Honeywell's RT-ARM are being used to manage the resource (i.e., CPU) contention of real-time operations on the aircraft. QuO is also being used to provide application adaptation in middleware for a dynamic collaborative planning application in the avionics platform. We scoped out five remaining tasks that BBN will help Boeing do under our remaining QuOIN effort to help WSOA get to the ground demonstration stage:
- Update WSOA to use QuO version 3
- Help add the interface (i.e., system condition objects) between the QuO middleware and the F15 network monitor
- Extend the QuO code generator to support a few additional identified features that WSOA requires
- Test the integrated WSOA QoS adaptive middleware, i.e., QuO/Dynamic Scheduler/RT ARM, and calibrate as necessary
- Help Boeing compile QuO and WSOA on the VxWorks platform

May 2001

During May, we worked on finishing up the last bit of packaging and debugging of QuOIN version 3 software and documentation, prepared the open-source license and began working on the distribution website.

We also prepared for participating in the Quorum PI meeting held in New Orleans, LA 22-25 May. We led a working group on Quorum program assessment, participated in the HiPer-D transition working group and prepared materials and information for both of these.

We also began the process of transitioning the QuO parts of the Boeing WSOA software to QuO version 3. We got the latest versions of WSOA code, began constructing a stand-alone test case at BBN and began porting the WSOA QuO code to QuO version 3.

Also during May, we continued working on the new AQuA gateway. We finished working on a testing phase of the new version of the AQuA system implemented in Solaris (new gateway with the active pass first and the passive handlers) using several applications: a simple pinger application, the "deet"

application and the castle application. We then started testing the Linux implementation. At the end of the period, the Solaris implementation and the active replication in the Linux implementation passed our list of tests. Passive replication in the Linux implementation still had some bugs that we are working to remove.

We also continued interacting with the HiPer-D Team on how to incorporate the AQuA framework into their 2001 demonstration. We interacted on the architecture of the AQuA part (presented at a high level during the PI meeting in New Orleans) and worked out a schedule (also presented during the PI meeting).

June 2001

The main milestone we reached in June was release of the QuOIN version 3 software. On June 1, we announced the first open-source release of the QuO and QuOIN software. The software is available at http://www.dist-systems.bbn.com/tech/QuO/quorelease.html.

QuO/QuOIN 3.0 has the following new features:
- Qoskets: mechanism for bundling QoS adaptive code for reuse
- C++ integrated QuO Kernel
- Java RMI support
- Aspect SDL (ASL):
  - Language independent (C++ and Java) with Java-like syntax
  - Multiple ASL files can be woven together
  - ASL Templates to indirect Method signatures
- State Machine Representation for Regions in CDL
- Resource Status Service: Unified architecture for monitoring the status of external resources from inside an application:
  - Default resource configurations published on web pages
  - QuO Status Typed Event Channel push technology for getting host load and capacity
  - Direct interface to Remos, for network monitoring
- Reusable in-band instrumentation, i.e., an instrumentation Qosket allows instrumentation independent of the Business interface
- Example Adaptive Code for:
  - Resource Monitoring with Client-side adaption (RSS and in-band instrumentation)
  - Bandwidth Reservation (RSVP)
  - Security (Authentication)
- Easy to install Linux RPMs

Also during June, we continued working on the new AQuA gateway. We finished the testing phase of the new version of the AQuA system implemented in Solaris and in Linux (new gateway with the active pass first and the passive handlers) using several applications: a simple pinger application, the "deet" application and the castle application. Both implementations passed our list of tests. We also worked towards the release of AQuA by writing a detailed user guide and by simplifying the installation and the execution of the AQuA system.

Also during the report period we attended the Quorum PI meeting in New Orleans, LA 22-25 May. We presented on the QuOIN efforts including information about the QuOIN 3 open-source release, the WSOA technology transition efforts and the NSWC technology transition efforts.

We also participated in the HiPer-D transition working group. We discussed and presented about the hardware and software requirements needed for a UAV demonstration in HiPer-D demo 2000. We identified the need for several workstations (preferably 5), running Linux (preferably Redhat 7.1) and some RSVP-enabled routers. We also discussed the hardware and software needs for, and plans for, transition of AQuA into HiPER-D demonstration 2001.

After the PI meeting, we continued to interact with the HiPER-D team. We scheduled two visits to NSWC, Dahlgren. The first, by the University of Illinois AQuA team, occurred on June 11 and 12. The goal of the visit was to install the new version of AQuA at NSWC and to customize AQuA for the application that will be used for the 2001 demonstration. We reached both goals by the end of the visit. Several components of the application still need to be developed and therefore require further interaction with us for a complete integration for the 2001 demonstration.

The second visit was by the BBN UAV team on June 25-26. The purpose of this visit to NSWCDD was to bring and install the current version of the UAV software, to discuss configuration issues and to plan the schedule for technology transition into HiPer-D demo 2001.

Shortly after the open-source release of QuO/QuOIN, we delivered a new release of code to Boeing for the WSOA effort. We built a standalone testcase for WSOA, with many of the Boeing, Scheduler and RTARM stubbed out, so that we could test the QuO part of WSOA at BBN. We upgraded the WSOA code in this example to work with QuO version 3, packaged up the updated application code and QuO version 3, and delivered it to Boeing on 13 June. We also provided Boeing with documentation, a detailed description of what changed and a description mapping components of the former (v2.1) WSOA to the new v3.0 pieces.

July 2001

We discussed the network configuration that NSWC will provide for the demonstration of the UAV software. NSWC will provide a network configuration that is capable of demonstrating the RSVP capability of the UAV application. Ideally this configuration will be provided in a non-secure lab, such as the DTL, to facilitate development and testing, as well as in the SCL. We discussed possible configurations of the network, the number and arrangement of routers, and the number and arrangements of subnets. We gathered information about NSWC's capabilities and discussed it with the networking and RSVP experts at BBN.

Also during July, we released a number of minor updates to the QuO software. Periodic updates with bug fixes and new functionality are available at our website at http://www.dist-systems.bbn.com/tech/QuO.

Also during July, we began to significantly enhance the AQuA framework. During this period, we have ported AQuA to NT, implemented a replicated dependability manager and simplified the way of installing and executing the AQuA system. The implementation of these features led to the decision to

delay the release until these features have been extensively tested (and maybe some other features added). The user guide that had been finalized when these new features were absent from the AQuA system will be revised accordingly.

August 2001

During the report period, we continued to provide support for the NSWC HiPer-D demo 2001 and Boeing's Weapon System Open Architecture (WSOA) effort, and the transition of QuOIN software to these programs.

We continued to develop the network management and middleware adaptation pieces of the UAV software and continued to interact with NSWCDD in preparation for a delivery for the HiPer-D 2001 demonstration. We attended a TAO workshop held in St. Louis, Missouri on 5-6 August. While in St. Louis, we met with OOMWorks, who has developed the CORBA A/V Streaming Service and AQoSA interface that QuO utilizes in the UAV software.

We also helped Boeing update the WSOA software for the WSOA quarterly review on 14 August. The update went smoothly, pointing out a few bugs in the QuO software that we quickly fixed. The capabilities of QuO version 3 in the WSOA application were demonstrated at Boeing's WSOA quarterly review on August 14.

We attended a workshop hosted by ONR-311 / KSAFNC and DARPA at Patuxent River on 23-24 August. At this workshop, we made a presentation about QuOIN technology, integration and transition activities, participated in discussions about the HiPer-D and WSOA projects and participated in discussions about possible interaction between these DARPA programs and Navy-sponsored programs.

We continued to make periodic releases of the QuO and QuOIN software via our website at http://www.dist-systems.bbn.com/tech/QuO/quorelease.html.

Also during August, we continued testing the various features of AQuA on the 3 different platforms: Solaris, Linux and NT. AQuA now includes a replicated dependability manager (with a new GUI for the dependability manager), active and passive replication (connection groups can now have a replication group using active replication and the other replication group using passive replication), a new advisor observer implementation and a simpler way to install and interface applications with AQuA. The detailed user guide has also been finalized and is now available.

September 2001

We continued to develop the network management and middleware adaptation pieces of the UAV software and continued to interact with NSWCDD in preparation for a delivery for the HiPer-D 2001 demonstration. We worked with NSWC helping to build and test RSVP in NSWC's network and to build ACE and TAO with AQoSA (RSVP) enabled. We did some further development of AQoSA, AVStreams and the UAV Connection_Manager, with the result being that we now have the capability to set RSVP reservations through a CORBA AVStreams interface and can receive information about changes in the RSVP reservations. We developed a QuO contract to coordinate the setup and maintenance of reservations along with the rest of the UAV adaptation. We performed some

experiments on the three-stage UAV application with network load and reservations. We plan to do some additional testing and experimenting with the UAV application using RSVP to determine the benefits of using network reservation.

After the August WSOA program review, we delivered a new version of the QuO software to Boeing for WSOA and proceeded to help Boeing port the QuO parts of WSOA to VxWorks for the upcoming flight demonstration. A few minor issues arose because of specific C++ features that weren't supported by VxWorks (e.g., namespaces), and ACE features that mapped to different things on VxWorks than on Linux and NT (e.g., the CORBA long long type). We fixed these, tested them and sent the patches to Boeing.

Also during September, we have released a new version of AQuA to BBN and NSWC. Some issues on an open source version need still to be addressed before releasing an open source version that a broader community of users might be interested in. We also have continued stress testing AQuA in order to improve its performance and to ease its usage.

We also have continued interacting with the HiPer-D team on how to incorporate the AQuA framework in their 2001 demonstration. The HiPer-D team confirmed that the incorporation of AQuA in new developed components of their demonstration was made simpler due to the new features offered by AQuA (e.g., use of improved scripts).

October 2001
During the report period, we continued to provide support for the NSWC HiPer-D demo 2001 and Boeing's Weapon System Open Architecture (WSOA) effort, and the transition of QuOIN software to these programs. We also kicked off efforts in the investigation of Differentiated Services (DiffServ) network management and Real-time CORBA.

We visited NSWC on 24-26 September where we installed the UAV software in both the DTL (non-secure laboratory) and the SCL (secure laboratory, where HiPer-D demonstrations are typically held). We also helped NSWC integrate the operation of the UAV software with their Program Control system.

The QuO software worked successfully with the NSWCDD Instrumentation Broker and the NSWCDD Instrumentation Display.

The following tests were executed:
A CPU load was placed on the host where Viewer 1 was executing. Viewer 1 started dropping frames, and went from contract region 12 to contract region 11. As expected, the video degraded. The CPU load was increased and Viewer 1 dropped more frames and went from contract region 11 to contract region 10. There was a further degradation of the video. Viewer 2 stayed in a high contract region, and the video did not degrade. The CPU load was decreased, and Viewer 1 dropped fewer frames and entered contract region 11. The quality of the video improved. The CPU load was lifted, Viewer 1 did not drop any frames, and entered region 12. The quality of the video improved.
The same test was executed where the load was placed on the host where Viewer 2 was executing with the same expected successful results.

A CPU load was placed on the hosts where Viewer 1 and Viewer 2 were executing. Viewer 1 and Viewer 2 started dropping frames and went from contract region 12 to contract region 11. Both of the videos degraded. The CPU load was increased on the hosts, and Viewer 1 and Viewer 2 dropped more frames and went from contract region 11 to contract region 10. Both videos degraded. The CPU load was decreased on the hosts and Video 1 and Video 2 dropped fewer frames, and entered contract region 11. The videos improved. The CPU load was lifted from the hosts, and Video 1 and Video 2 entered contract region 12. Video 1 and Video 2 did not drop and frames, and the quality of the videos improved.

A CPU load was placed on the host where the Distributor was executing. Viewer 1 and Viewer 2 went from contract region 12 to contract region 11 because of the reduced frame rate. Both videos degraded. The CPU load was increased on the Distributor host, and Viewer 1 and Viewer 2 went from contract region 11 to contract region 10 because of the further reduced frame rate. The videos degraded. The CPU load was decreased on the Distributor host, and Viewer 1 and Viewer 2 and entered contract region 11 because of the increase in frame rate. The videos improved. The CPU load was lifted from the Distributor host, and Viewer 1 and Viewer 2 entered contract region 12 because of the increase in frame rate. The videos improved.

Another Distributor was executed on a different host than the original Distributor was running on. The new Distributor started sending video data to Video 1 and Video 2. The original Distributor was killed, and Video 1 and Video continued sending data to Video 1 and Video 2.

The software that we delivered to NSWC includes network management capability using RSVP that they were not ready to demonstrate in the Hiper-D demo 2001. They are targeting demonstration of those capabilities later in the demo cycle, when they have the proper testbed equipment, including routers, and have settled on a design for the integration with the NSWC resource manager.

In lieu of this, we constructed a suitable testbed at BBN and fully tested and experimented with the RSVP capabilities in this testbed. This demonstration, which includes three receivers: one that is critical and therefore gets a network reservation sufficient to maintain 30 frames per second of video, a second that is less critical and uses frame dropping to keep the video current when the network is loaded and a control video that shows what the video would look like under load with no adaptation - is going to be demonstrated as part of the PCES PI meeting at the end of October in Mesa, AZ. (We are using aspect language technology being developed under BBN's PCES contract, AIRES, to implement some of the UAV capabilities.)

We kicked off efforts to investigate the use of Differentiated Services (DiffServ) and Real-Time CORBA during the current report period. We began planning additions to the AVStreams service to support DiffServ. We also began looking at the implementations of RT CORBA that OOMWorks has been working on, plan activities to evaluate it and get it working and plan development activities to utilize it.

We continued to make periodic releases of the QuO and QuOIN software via our website at http://www.dist-systems.bbn.com/tech/QuO/quorelease.html.

We completed all AQuA related activities at the end of September.

November 2001

During the report period, we continued investigating Differentiated Services (DiffServ) network management and Real-time CORBA, periodic open-source releases of the QuOIN software and supporting the technology transition efforts to Boeing's WSOA program and to NSWC's HiPer-D demonstration. We also kicked off an effort to use BBN's resource status service (RSS) to get network resource information for the purpose of making decisions about the types of adaptation that are appropriate to maintain the QoS of the UAV video streams.

DiffServ Network Management. We got a preliminary version of the CORBA A/V Streams Service supporting DiffServ to work. We modified AVStreams to enable setting of DiffServ codepoints. We tested this by setting DiffServ codepoints using the A/V Streams service through the passing of parameters to the A/V Stream Control Endpoints. Then we used the "modify_QoS" interface on AVStreams and verified that the Diffserv Codepoint values of the UDP/IP packets were changed appropriately.

We did some further testing using the new DiffServ support in AVStreams in conjunction with the Linux traffic control utility to achieve noticeable change in network behavior. We tested this by starting up the UAV application with the sender, distributor and receiver on three different hosts, then starting additional receivers without any traffic control. After starting 5 receivers, there was perceptible and comparable loss in data on all the receivers. Then we set up the effective forwarding and best effort on the network interface of the distributor. This put constraints on the best effort traffic and still exhibited perceptible and comparable loss in data on all the receivers. We then set the DiffServ codepoints for effective forwarding on selected streams and saw that they immediately performed better than the best effort traffic.

Real-time CORBA. During the report period, we began exploring how we can use the capabilities of Real-Time CORBA to preserve end-to-end QoS guarantees in the UAV application. Our plan is to explore the following two capabilities: (1) how the RT-CORBA priority of a scheduled task can be preserved as the task moves between nodes in a distributed system; and (2) how we can exploit the Dynamic Scheduling capabilities of RT-CORBA to use as an adaptation mechanism in response to various loads and threat conditions.

We designed an example application to test the capabilities of RT CORBA prior to inserting it into the UAV application. This example tests the following RT CORBA features:
- PriorityBandedConnectionPolicy, PriorityBands
- threadpool, threadpool with lanes
- PriorityModelPolicy
- ServerProtocolPolicy
- ClientProtocolPolicy
- PrivateConnectionPolicy
- create_reference_with_priority et. al

Resource Status to Control Adaptation. The current QuO UAV application has three levels of adaptation: adaptation within the application (frame dropping/data filtering), network-level adaptation

(RSVP reservations) and load-balancing adaptation (distributor re-hosting). The decision to execute any of these adaptive mechanisms in the QuO UAV application being demonstrated by NSWC is determined by application level measures of performance (i.e., frame rate). The QuO contract adaptation decision engine can make a better decision of what type of adaptation to employ if it has current information about the available resources in the system e.g., whether enough bandwidth is available to make a reservation. The QuO Resource Status Service (RSS) is a suite of tools that can provide status information about the network and hosts across a distributed system. It can provide information, such as network bandwidth available, network load and host load to make better adaptive decisions, especially at the level of network adaptation.

During the report period, we kicked off an effort to integrate the RSS with the current toolkits that we use for network level adaptation: AQoSA and AVStreams. This effort will involve developing the necessary interfaces into AQoSA and AVStreams to integrate the RSS information into the intelligent decision making capability of network-level adaptations. For example, when setting up an RSVP reservation using AQoSA and AVStreams, it would be highly desirable for AQoSA to query the network bandwidth available, and use that information to generate the parameters for an RSVP reservation request.

Other activities. We released QuO/QuOIN 3.0.8 as open-source during the report period via our website at http://www.dist-systems.bbn.com/tech/QuO/quorelease.html. This version fixed some bugs and included some improvements, including a new example similar in some respects to the WSOA application and the original Bette example, but using a webcam to provide the images, enhance JacORB support and improvements to the Qosket component features of QuO.

We demonstrated the UAV application at the PCES PI meeting as part of our AIRES activity under PCES. We have added some of the QoS control and adaptation in the UAV application using technology – including the QuO AOP languages – developed under the AIRES project. The demonstration focused on the software engineering and AOP technology used to program the capabilities, but also included all of the features developed under the QuOIN contract and demonstrated in the NSWC HiPer-D demo 2001. It also included the network reservation capabilities using RSVP that is fully functional in the NSWC software but was not included in their demonstration scenario.

December 2001

During the report period, we continued to work on technical efforts for the QuOIN contract, with the intention of wrapping them up during the next report period in anticipation of the contract's end. We continued investigating Differentiated Services (DiffServ) network management and Real-time CORBA, continued efforts to integrate BBN's resource status service (RSS) with the UAV software, continued periodic open-source releases of the QuOIN software and continued support of technology transition efforts to Boeing's WSOA program and to NSWC's HiPer-D demonstration.

DiffServ Network Management. We wrapped up our QuOIN efforts on the investigation of DiffServ Network Management by further testing the A/V Streams methods that we developed that enable the setting of DiffServ codepoints. We verified that this was working using the Ethereal protocol analyzer and by visual confirmation of improvement in the video. We ran experiments using both FreeBSD ALTQ (alternative queuing) and Linux traffic control.

Real-time CORBA. During the last report period, we had developed a test case and ran a set of experiments to verify that the Real-Time CORBA (RT CORBA) implementation that is included in TAO 1.2 works satisfactorily. The results of these experiments indicated that TAO's RT CORBA was behaving as expected, so we proceeded to use it as our Real-Time CORBA implementation in an attempt to integrate RT CORBA features into the UAV example. We concentrated on the priority management facilities in RT CORBA since they represent the core functionality that it offers.

To properly demonstrate RT CORBA in the UAV context we required two main system attributes: prioritized tasks, with associated thread priorities for performing the tasks, and scarce resources, typically CPU, that the tasks access using CORBA. However, the existing UAV application was not composed of distinctive tasks, and it was single threaded, so the first attribute was not present, and the core communication was done using AVStreams/UDP, not CORBA, so the second attribute wasn't present either.

To satisfy the first requirement we separated the per-receiver frame dispatching which occurs in the distributor process into separate tasks and introduced a worker thread per task to perform the work, plus a mechanism for setting each thread's priority. To satisfy the second requirement we introduced a simulated CPU-intensive frame processing service that is invoked via CORBA by the frame-dispatching task for each frame.

We introduced the new behaviors into the distributor using two QuO delegates. The first handles the queuing of frames per receiver, creates the worker thread to do frame dispatching and manages the priority of the worker thread. The priority of the worker thread is maintained by a specialized QuO system condition object that updates the worker thread's priority when it's value is changed. The second delegate introduces a call out to the frame processor object. The delegates both wrap the same interface and can be selected or deselected along with various other per-frame activities at runtime by chaining delegates together in different orders. For example, frame processing can be introduced before frame filtering, after frame filtering or both. By using delegates we were able to introduce the new system attributes with minimal impact on the base distributor code.

We introduced the frame processing simulator as a separate process. The CORBA object that it implements returns each frame it is passed without modification after performing a configurable amount of busy work on it. The work performed is $O(m*n*log_2n)$, where m is the configuration constant and n is the size of the frame. The CORBA object is configured to use client propagated priority so that the servant thread in the frame processing simulator inherits the priority of the worker thread in the distributor. Without this policy the work being done on behalf of a high priority distributor task would have to compete with work done on behalf of lower priority distributor tasks, or it could be forced to wait for a medium priority system task, leading to priority inversion.

Technology Transition. We continued to provide occasional support to Boeing in development of the WSOA flight and ground demonstration code. In support of the transition to the NSWC HiPer-D testbed, we received a brief report from Shafqat Anwar of the University of Texas, Arlington, describing a high level set of requirements for the integration with the NSWC resource manager. This was a follow-up to the discussions that we had with Lonnie Welch.

Other activities. We released QuO/QuOIN 3.0.9 as open-source during the report period via our website at http://www.dist-systems.bbn.com/tech/QuO/quorelease.html.

January 2002

During the report period, we continued to work on technical efforts for the QuOIN contract, with the intention of wrapping them up and begin producing the final set of contract deliverables. We wrapped up our investigation of Real-time CORBA and integration of BBN's resource status service (RSS) with the UAV software. We also continued our periodic open-source releases of the QuOIN software and continued support of technology transition efforts to Boeing's WSOA program and to NSWC's HiPer-D demonstration. Finally, we began producing the documentation for the final set of contract deliverables.

Real-time CORBA. We previously described two modifications that we made to the UAV distributor functionality to support the integration of Real-Time CORBA. First, we separated the per-receiver frame dispatching which occurs in the distributor process into separate tasks and introduced a worker thread per task to perform the work, plus a mechanism for setting each thread's priority. Second, we introduced a simulated CPU-intensive frame processing service that is invoked via CORBA by the frame-dispatching task for each frame. We added both of these behaviors using QuO delegates.

With these in place, we were able to use Real-time CORBA to manage the priorities and priority propagation of tasks in the application. We tested this by modifying the value of a worker priority system condition on the distributor and observing (via instrumentation) that both the worker thread and the frame processing simulator thread were executing with the new priority, as expected. Additional work needs to be done to verify that priorities are properly reflected at the operating system level, and thread pools and connection management need to be introduced, but these should be straightforward.

Resource Status to Control Adaptation. We wrapped up the integration of RSS into the adaptive middleware to provide information useful in making adaptive decisions, building upon the progress we had made previously. We added a method to enable an endpoint to be registered with an RSS client. When it is poked, the RSS client will update a property with the currently available bandwidth. The AV Streams flow handler then looks at the available bandwidth property to determine how large a reservation is feasible before it makes a reservation.

8. Publications, Presentations, Demonstrations, Educational and Idea Dissemination Activities

Project Publications

Schantz R.E., Schmidt D.C., "Middleware for Distributed Systems - Evolving the Common Structure for Network-centric Applications." Chapter in the Encyclopedia of Software Engineering; John Wiley & Sons, December 2001

Schmidt D.C., Schantz R.E., Masters M.W., Cross J.K., Sharp D.C., DiPalma L.P., "Toward Adaptive and Reflective Middleware for Network-Centric Combat Systems," Crosstalk The Journal of Defense Software Engineering, November 2001, pp. 10-16

Loyall J.P., Schantz R.E., Pal P., Zinky J., Atighetchi M., "Emerging Patterns in Adaptive, Distributed Real-Time, Embedded Middleware," OOPSLA 2001 Workshop - Towards Patterns and Pattern Languages for OO Distributed Real-time and Embedded Systems, October 14, 2001, Tampa Bay, Florida

Karr D.A., Rodrigues C., Loyall J.P., Schantz R.E., "Controlling Quality-of-Service in a Distributed Video Application by an Adaptive Middleware Framework," Proceedings of ACM Multimedia 2001, September 30 - October 5, 2001, Ottawa, Ontario, Canada

Karr D.A., Rodrigues C., Loyall J.P., Schantz R.E., Krishnamurthy Y., Pyarali I., Schmidt D.C., "Application of the QuO Quality-of-Service Framework to a Distributed Video Application," Proceedings of the International Symposium on Distributed Objects and Applications, September 18-20, 2001, Rome, Italy

Cukier M., Lyons J., Pandey P., Ramasamy H.V., Sanders W.H., Pal P., Webber F., Schantz R., Loyall J., Watro R., Atighetchi M., Gossett J., "Intrusion Tolerance Approaches in ITUA," Fast Abstract in Supplement of the 2001 International Conference on Dependable Systems and Networks, July 1-4, 2001, Göteborg, Sweden, pp. B-64 to B-65

Krishnamurthy Y., Kachroo V., Karr D.A., Rodrigues C., Loyall J.P., Schantz R.E., Schmidt D.C., "Integration of QoS-Enabled Distributed Object Computing Middleware for Developing Next-Generation Distributed Applications," Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), June 18, 2001, Snowbird, Utah

Loyall J.L., Gossett J.M., Gill C.D., Schantz R.E., Zinky J.A., Pal P., Shapiro R., Rodrigues C., Atighetchi M., Karr D., "Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications," Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS-21), April 16-19, 2001, Phoenix, Arizona

Rodrigues C., Gill C.D., "Controlling a CORBA Real-time Event Channel with QuO Middleware," Unpublished

Schantz R.E., Zinky J.A., Loyall J.P., Shapiro R., Megquier J., "Adaptable Binding for Quality of Service in Highly Networked Applications," Proceedings of SSGRR-2000, International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, July 31-August 6, 2000, L'Aquila, Italy

Rodrigues C., Loyall J.P., Schantz R.E., "Quality Objects (QuO): Adaptive Management and Control Middleware for End-to-End QoS," OMG's First Workshop on Real-Time and Embedded Distributed Object Computing, July 24-27, 2000, Falls Church, Virginia, USA

Pal P.P., Loyall J.P., Schantz R.E., Zinky J.A., Shapiro R., Megquier J., "Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration," Proceedings of ISORC 2000, The Third IEEE International Symposium on Object-Oriented Real-time Distributed Computing, March 15-17, 2000, Newport Beach, CA

Loyall J.P., Pal P.P., Schantz R.E., Webber F., "Building Adaptive and Agile Applications Using Intrusion Detection and Response," Proceedings of NDSS 2000, the Network and Distributed System Security Symposium, February 2-4 2000, San Diego, CA

Schantz, R.E., "Quality of Service," a survey article prepared for Encyclopedia of Distributed Computing, Kluwer Academic Publishers, Partha Dasgupta and Joseph Urban, editors, to appear 1999

Ren J., Cukier M., Rubel P., Sanders W.H., Bakken D.E., Karr D.A., "Building Dependable Distributed Applications Using AquA," Proceedings of the 4th IEEE Symposium on High Assurance Systems Engineering (HASE'99), November 17-19, 1999, Washington D.C., pp. 189-196

Cukier M., Ren J., Rubel P., Bakken D.E., Karr D.A., "Building Dependable Distributed Objects with the AquA Architecture," Digest of FastAbstracts presented at the 29th Annual International Symposium on Fault-Tolerant Computing (FTCS-29), June 15-18, 1999, Madison, Wisconsin, pp. 17-18

Schantz R., Zinky J.A., Karr D.A., Bakken D.E., Megquier J., Loyall J.P., "An Object-level Gateway Supporting Integrated-Property Quality of Service," Proceedings of ISORC '99, The 2nd IEEE International Symposium on Object-oriented Real-time distributed Computing, May 2-5, 1999, Palais du Grand Large 35 407 Saint-Malo, FRANCE

Sabnis C., Cukier M., Ren J., Rubel P., Sanders W.H., Bakken D.E., Karr D.A., "Proteus: A Flexible Infrastructure to Implement Adaptive Fault Tolerance in AQuA," in C. B. Weinstock and J. Rushby (Eds.), Dependable Computing for Critical Applications 7, vol. 12 in series Dependable Computing and Fault-Tolerant Systems (A. Avizienis, H.

Kopetz, and J. C. Laprie, Eds.), pp. 149-168. Los Alamitos, CA: IEEE Computer Society, 1999

Cukier M., Ren J., Sabnis C., Henke D., Pistole J., Sanders W.H., Bakken D.E., Berman M.E., Karr D.A., Schantz R.E., "AQuA: An Adaptive Architecture That Provides Dependable Distributed Objects," Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems (SRDS'98), October 20-23, 1998, West Lafayette, Indiana, pp. 245-253

Vanegas R., Zinky J.A., Loyall J.P., Karr D.A., Schantz R.E., Bakken D.E., "QuO's Runtime Support for Quality of Service in Distributed Objects," Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), 15-18 September 1998, The Lake District, England

Schantz, R.E., "BBN and the Defense Advanced Research Projects Agency," Prepared as a Case Study for America's Basic Research: Prosperity Through Discovery, A Policy Statement by the Research and Policy Committee of the Committee for Economic Development (CED), June 1998

Loyall J.P., Bakken D.E., Schantz R.E., Zinky J.A., Karr D.A., Vanegas R., Anderson K.R., "QoS Aspect Languages and Their Runtime Integration," Lecture Notes in Computer Science, Vol. 1511, Springer-Verlag. Proceedings of the Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR98), 28-30 May 1998, Pittsburgh, Pennsylvania

Loyall J.P., Schantz R.E., Zinky J.A., Bakken D.E., "Specifying and Measuring Quality of Service in Distributed Object Systems," Proceedings of the First International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '98), 20-22 April 1998, Kyoto, Japan

H. S. Duggal, M. Cukier, and W. H. Sanders, "Probabilistic Verification of a Synchronous Round-based Consensus Protocol," *Proceedings of the Sixteenth IEEE Symposium on Reliable Distributed Systems (SRDS-97),* Durham, North Carolina, October 22-24, 1997, pp. 165-174

S. Krishnamurthy, W. H. Sanders, and M. Cukier, "A Dynamic Replica Selection Algorithm for Tolerating Timing Faults," *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2001)*, Göteborg, Sweden, July 1-4, 2001, pp. 107-116

S. Krishnamurthy, W. H. Sanders, and M. Cukier, "An Adaptive Framework for Tunable Consistency and Timeliness Using Replication," to appear in *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2002)*, Washington D.C., USA

S. Krishnamurthy, W. H. Sanders, and M. Cukier, "Performance Evaluation of a Probabilistic Replica Selection Algorithm," *Proceedings of the 7th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2002)*, San Diego, California, January 7-9, 2002

Y. (J.) Ren, T. Courtney, M. Cukier, C. Sabnis, W. H. Sanders, M. Seri, D. A. Karr, P. Rubel, R. E. Schantz, and D. Bakken, "AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects," *IEEE Transactions on Computers*, to appear.
J. Ren, M. Cukier, P. Rubel, W. H. Sanders, D. E. Bakken, and D. A. Karr, "Building Dependable Distributed Applications Using AQuA," *Proceedings of the 4th IEEE International Symposium on High-Assurance Systems Engineering (HASE '99)*, Washington, D.C., November 17-19, 1999, pp. 189-196

Y. Ren, M. Cukier, and W. H. Sanders, "An Adaptive Algorithm for Tolerating Value Faults and Crash Failures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 2, Feb. 2001, pp. 173-192

W. H. Sanders, "Building Dependable Distributed Systems Using the AQuA Architecture" (invited talk) *Proceedings of SCTF'2001 - IX Brazilian Symposium on Fault-Tolerant Computing*, Florianópolis, Santa Catarina, Brazil, March 5-7, 2001, p. 1.
M. Seri, T. Courtney, M. Cukier, and W. H. Sanders, "An Overview of the AQuA Gateway," *Proceedings of the 1st Workshop on The ACE ORB (TAO)*, St. Louis, MO, August 5-6, 2001, to appear

R. Chandra, M. Cukier, R. M. Lefever, and W. H. Sanders, "Dynamic Node Management and Measure Estimation in a State-Driven Fault Injector," *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS 2000)*, Nürnberg, Germany, October 16-18, 2000, pp. 248-257

R. Chandra, R. M. Lefever, M. Cukier, and W. H. Sanders, "Loki: A State-Driven Fault Injector for Distributed Systems," *Proceedings of the International Conference on Dependable Systems and Networks (DSN-2000)*, New York, NY, June 25-28, 2000, pp. 237-242

R. Chandra, R. M. Lefever, K. Joshi, M. Cukier, and W. H. Sanders, "A Global-State-Triggered Fault Injector for Distributed System Evaluation," not yet published.
M. Cukier, R. Chandra, D. Henke, J. Pistole, and W. H. Sanders, "Fault Injection Based on a Partial View of the Global State of a Distributed System," *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems (SRDS)*, Lausanne, Switzerland, October 19-22, 1999, pp. 168-177

Theses

R. U.V. Chandra, *Loki: A State-Driven Fault Injector for Distributed Systems*, Master's thesis, University of Illinois at Urbana-Champaign, 2001

D. Henke, *Loki–An Empirical Evaluation Tool for Distributed Systems: The Experiment Analysis Framework*, Master's thesis, University of Illinois at Urbana-Champaign, 1998.
S. Krishnamurthy, *An Adaptive Quality of Service Aware Middleware for Accessing Replicated Objects*, Ph.D. thesis, University of Illinois at Urbana-Champaign, 2002, to appear

J. Pistole, *Loki–An Empirical Evaluation Tool for Distributed Systems: The Run-Time Experiment Framework*, Master's thesis, University of Illinois at Urbana-Champaign, 1998

Y. Ren, *AQuA: A Framework for Providing Adaptive Fault Tolerance to Distributed Applications*, Ph.D. thesis, University of Illinois at Urbana-Champaign, 2001.
P. G. Rubel, *Passive Replication in the AQuA System*, Master's thesis, University of Illinois at Urbana-Champaign, 2000

B. Sabnis, *Proteus: A Software Infrastructure Providing Dependability for CORBA Applications*, Master's thesis, University of Illinois at Urbana-Champaign, 1998